Iris Proof Mode

Interactive Proofs in Higher-Order Concurrent Separation Logic

Robbert Krebbers[1]

Delft University of Technology, The Netherlands

June 12, 2017 @ MFPS, Ljubljana, Slovenia

[1]Iris is joint work with: Ralf Jung, Jacques-Henri Jourdan, Aleš Bizjak, Hoang-Hai Dang, Jan-Oliver Kaiser, David Swasey, Filip Sieczkowski, Kasper Svendsen, Aaron Turon, Amin Timany, Derek Dreyer, and Lars Birkedal

Goal of this talk

Many recent program logics come with mechanized soundness proofs, but how to reason in these logics?

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

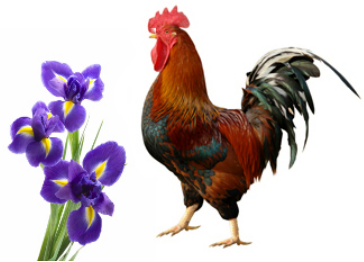
Goal of this talk

Many recent program logics come with mechanized soundness proofs, but how to reason in these logics?

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

How?

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



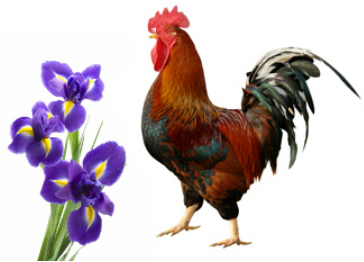
Goal of this talk

Many recent program logics come with mechanized soundness proofs, but how to reason in these logics?

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

How?

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



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

Iris Proof Mode (IPM) demo

```
Lemma and_exist_sep {A} P R (Ψ: A → iProp) :
  P ∗ (∃ a, Ψ a) ∗ R -∗ ∃ a, Ψ a ∗ P.
Proof.
```

```
1 subgoal
M : ucmraT
A : Type
P, R : iProp
Ψ : A → iProp
_____________________________________(1/1)
P ∗ (∃ a : A, Ψ a) ∗ R -∗ ∃ a : A, Ψ a ∗ P
```

Iris Proof Mode (IPM) demo

```
Lemma and_exist_sep {A} P R (Ψ: A → iProp) :
  P ∗ (∃ a, Ψ a) ∗ R -∗ ∃ a, Ψ a ∗ P.
Proof.
  iIntros "[HP [HΨ HR]]".
```

```
1 subgoal
M : ucmraT
A : Type
P, R : iProp
Ψ : A → iProp
_____________________________________(1/1)
P ∗ (∃ a : A, Ψ a) ∗ R -∗ ∃ a : A, Ψ a ∗ P
```

Iris Proof Mode (IPM) demo

```
Lemma and_exist_sep {A} P R (Ψ: A → iProp) :
  P ∗ (∃ a, Ψ a) ∗ R -∗ ∃ a, Ψ a ∗ P.
Proof.
  iIntros "[HP [HΨ HR]]".
```

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

Iris Proof Mode (IPM) demo

```
Lemma and_exist_sep {A} P R (Ψ: A → iProp) :
  P ∗ (∃ a, Ψ a) ∗ R -∗ ∃ a, Ψ a ∗ P.
Proof.
  iIntros "[HP [HΨ HR]]".
  iDestruct "HΨ" as (x) "HΨ".
```

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

Iris Proof Mode (IPM) demo

```
Lemma and_exist_sep {A} P R (Ψ: A → iProp) :
  P ∗ (∃ a, Ψ a) ∗ R -∗ ∃ a, Ψ a ∗ P.
Proof.
  iIntros "[HP [HΨ HR]]".
  iDestruct "HΨ" as (x) "HΨ".
```

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

Iris Proof Mode (IPM) demo

```
Lemma and_exist_sep {A} P R (Ψ: A → iProp) :
  P ∗ (∃ a, Ψ a) ∗ R -∗ ∃ a, Ψ a ∗ P.
Proof.
  iIntros "[HP [HΨ HR]]".
  iDestruct "HΨ" as (x) "HΨ".
  iExists x.
```

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

Iris Proof Mode (IPM) demo

```
Lemma and_exist_sep {A} P R (Ψ: A → iProp) :
  P ∗ (∃ a, Ψ a) ∗ R -∗ ∃ a, Ψ a ∗ P.
Proof.
  iIntros "[HP [HΨ HR]]".
  iDestruct "HΨ" as (x) "HΨ".
  iExists x.
```

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

Iris Proof Mode (IPM) demo

```
Lemma and_exist_sep {A} P R (Ψ: A → iProp) :
  P ∗ (∃ a, Ψ a) ∗ R -∗ ∃ a, Ψ a ∗ P.
Proof.
  iIntros "[HP [HΨ HR]]".
  iDestruct "HΨ" as (x) "HΨ".
  iExists x.
  iSplitL "HΨ".
```

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

Iris Proof Mode (IPM) demo

```
Lemma and_exist_sep {A} P R (Ψ: A → iProp) :
  P ∗ (∃ a, Ψ a) ∗ R -∗ ∃ a, Ψ a ∗ P.
Proof.
  iIntros "[HP [HΨ HR]]".
  iDestruct "HΨ" as (x) "HΨ".
  iExists x.
  iSplitL "HΨ".
```

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

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

Iris Proof Mode (IPM) demo

```
Lemma and_exist_sep {A} P R (Ψ: A → iProp) :
  P ∗ (∃ a, Ψ a) ∗ R -∗ ∃ a, Ψ a ∗ P.
Proof.
  iIntros "[HP [HΨ HR]]".
  iDestruct "HΨ" as (x) "HΨ".
  iExists x.
  iSplitL "HΨ".
  -
```

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

Iris Proof Mode (IPM) demo

```
Lemma and_exist_sep {A} P R (Ψ: A → iProp) :
  P ∗ (∃ a, Ψ a) ∗ R -∗ ∃ a, Ψ a ∗ P.
Proof.
  iIntros "[HP [HΨ HR]]".
  iDestruct "HΨ" as (x) "HΨ".
  iExists x.
  iSplitL "HΨ".
  - iAssumption.
```

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

Iris Proof Mode (IPM) demo

```
Lemma and_exist_sep {A} P R (Ψ: A → iProp) :
  P ∗ (∃ a, Ψ a) ∗ R -∗ ∃ a, Ψ a ∗ P.
Proof.
  iIntros "[HP [HΨ HR]]".
  iDestruct "HΨ" as (x) "HΨ".
  iExists x.
  iSplitL "HΨ".
  - iAssumption.
```

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

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

Iris Proof Mode (IPM) demo

```
Lemma and_exist_sep {A} P R (Ψ: A → iProp) :
  P ∗ (∃ a, Ψ a) ∗ R -∗ ∃ a, Ψ a ∗ P.
Proof.
  iIntros "[HP [HΨ HR]]".
  iDestruct "HΨ" as (x) "HΨ".
  iExists x.
  iSplitL "HΨ".
  - iAssumption.
  -
```

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

Iris Proof Mode (IPM) demo

```
Lemma and_exist_sep {A} P R (Ψ: A → iProp) :
  P ∗ (∃ a, Ψ a) ∗ R -∗ ∃ a, Ψ a ∗ P.
Proof.
  iIntros "[HP [HΨ HR]]".
  iDestruct "HΨ" as (x) "HΨ".
  iExists x.
  iSplitL "HΨ".
  - iAssumption.
  - iAssumption.
```

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

Iris Proof Mode (IPM) demo

```
Lemma and_exist_sep {A} P R (Ψ: A → iProp) :      No more subgoals.
  P ∗ (∃ a, Ψ a) ∗ R −∗ ∃ a, Ψ a ∗ P.
Proof.
  iIntros "[HP [HΨ HR]]".
  iDestruct "HΨ" as (x) "HΨ".
  iExists x.
  iSplitL "HΨ".
  - iAssumption.
  - iAssumption.
```

Iris Proof Mode (IPM) demo

```
Lemma and_exist_sep {A} P R (Ψ: A → iProp) :      No more subgoals.
  P ∗ (∃ a, Ψ a) ∗ R −∗ ∃ a, Ψ a ∗ P.
Proof.
  iIntros "[HP [HΨ HR]]".
  iDestruct "HΨ" as (x) "HΨ".
  iExists x.
  iSplitL "HΨ".
  - iAssumption.
  - iAssumption.
Qed.
```

Iris Proof Mode (IPM) demo

```
Lemma and_exist_sep {A} P R (Ψ: A → iProp) :
  P ∗ (∃ a, Ψ a) ∗ R -∗ ∃ a, Ψ a ∗ P.
Proof.
  iIntros "[HP [HΨ HR]]".
  iDest
  iExis
  iSplitL "HΨ".
  - iAssumption.
  - iAssumption.
Qed.
```

Logical notations overridden in scope for Iris

Iris Proof Mode (IPM) demo

```
Lemma and_exist_sep {A} P R (Ψ: A → iProp) :
  P ∗ (∃ a, Ψ a) ∗ R -∗ ∃ a, Ψ a ∗ P.
Proof.
  iIntros "[HP [HΨ HR]]".
```

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

Notation for deeply embedded context

Iris Proof Mode (IPM) demo

```
Lemma and_exist_sep {A} P R (Ψ: A → iProp) :
  P ∗ (∃ a, Ψ a) ∗ R -∗ ∃ a, Ψ a ∗ P.
Proof.
  iIntros "[HP [HΨ HR]]".
  Unset Printing Notations.
```

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

Notation for deeply embedded context

Iris Proof Mode (IPM) demo

```
Lemma and_exist_sep {A} P R (Ψ: A → iProp) :
  P ∗ (∃ a, Ψ a) ∗ R −∗ ∃ a, Ψ a ∗ P.
Proof.
  iIntros "[HP [HΨ HR]]".
  Unset Printing Notations.
```

```
1 subgoal
M : ucmraT
A : Type@{Top.105}
P, R : uPred M
Ψ : forall _ : A, uPred M
____________________________________(1/1)
@uPred_entails M
 (@of_envs M
   (@Envs M (@Enil (uPred M))
    (@Esnoc (uPred M)
      (@Esnoc (uPred M)
       (@Esnoc (uPred M) (@Enil (uPred M))
         (String
          (Ascii false false false true false
     false true
            false)
           (String
             (Ascii false false false false true
      false true
              false) EmptyString)) P)
       (String
```

Motivation

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

Infeasible to automate everything, for example:

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

Most of these projects are formalized in IPM

How to do such proofs in a proof assistant?

Current proof assistant support is limited to basic separation logic:

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

Iris has many complicated connectives that are beyond basic separation logic:

- Various modalities, e.g., $\triangleright$, $\Box$, $\Rightarrow\!\!\!\!\mid$
- Guarded recursion and Löb induction
- Heavy use of magic wand $-\!\!*$
- Non-trivial use of higher-order quantification
- Ghost ownership
- Impredicative invariants

How to embed a logic into a proof assistant

Deep embedding

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

Shallow embedding

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

How to embed a logic into a proof assistant

Deep embedding

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

Traverse formulas using Coq functions (fast)

Reflective tactics (fast)

Shallow embedding

```
Definition iProp : Type :=
  (* 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)

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 like lists

Shallow embedding

```
Definition iProp : Type :=
  (* 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 like 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 like lists

Grammar of formulas fixed once and forall

Shallow embedding

```
Definition iProp : Type :=
  (* 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 like lists from Coq

Easily extensible with new connectives

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 like lists

Grammar of formulas fixed once and forall

Shallow embedding

```
Definition iProp : Type :=
  (* 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 like lists from Coq

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 in IPM

Visible goal in IPM:

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

---

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

---□

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

---*

R Goal in object logic

Deeply embedded contexts in IPM

Visible goal in IPM:

Propositions that enjoy $P \Leftrightarrow P * P$

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

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

---□

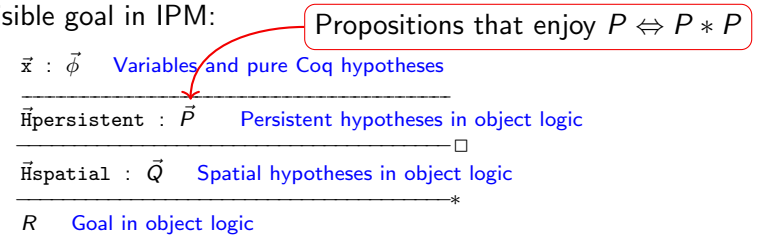
$\vec{H}$`spatial` $: \vec{Q}$ Spatial hypotheses in object logic

---*

R Goal in object logic

Deeply embedded contexts in IPM

Visible goal in IPM:



Actual Coq goal (without pretty printing):

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

---

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

where:

```
Record envs :=
  Envs { env_persistent : env iProp; env_spatial : env iProp }.
Coercion of_envs (Δ : envs) : iProp :=
  (⌜ envs_wf Δ ⌝ ∗ □ [∗] env_persistent Δ ∗ [∗] env_spatial Δ)%I.
```

Deeply embedded contexts in IPM

Visible goal in IPM:

Propositions that enjoy $P \Leftrightarrow P * P$

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

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

---□

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

---*

R Goal in object logic

Actual Coq goal (without pretty printing):

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

$\texttt{of\_envs}\ (\texttt{Envs}\ \ \ldots\ \ \ldots) \vdash R$

where:

Association list of shallowly embedded propositions

```
Record envs :=
  Envs { env_persistent : env iProp; env_spatial : env iProp }.
Coercion of_envs (Δ : envs) : iProp :=
  (⌜ envs_wf Δ ⌝ ∗ □ [∗] env_persistent Δ ∗ [∗] env_spatial Δ)%I.
```

Deeply embedded contexts in IPM

Visible goal in IPM:

Propositions that enjoy $P \Leftrightarrow P * P$

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

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

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

R Goal in object logic

Actual Coq goal (without pretty printing):

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

$\texttt{of\_envs}\ (\texttt{Envs}\ \ldots\ \ldots) \vdash R$

where:

Association list of shallowly embedded propositions

Folded separating conjunction

```
Record envs :=
  Envs { env_persistent : env iProp; env_spatial : env iProp }.
Coercion of_envs (Δ : envs) : iProp :=
  (⌜envs_wf Δ⌝ ∗ □ [∗] env_persistent Δ ∗ [∗] env_spatial Δ)%I.
```

The iSplit tactic

```
Lemma and_exist_sep {A} P R (Ψ: A → iProp) :
  P ∗ (∃ a, Ψ a) ∗ R -∗ ∃ a, Ψ a ∗ P.
Proof.
  iIntros "[HP [HΨ HR]]".
  iDestruct "HΨ" as (x) "HΨ".
  iExists x.
```

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

The iSplit tactic

```
Lemma and_exist_sep {A} P R (Ψ: A → iProp) :
  P ∗ (∃ a, Ψ a) ∗ R −∗ ∃ a, Ψ a ∗ P.
Proof.
  iIntros "[HP [HΨ HR]]".
  iDestruct "HΨ" as (x) "HΨ".
  iExists x.
  iSplitL "HΨ".
```

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

The iSplit tactic

```
Lemma and_exist_sep {A} P R (Ψ: A → iProp) :
  P ∗ (∃ a, Ψ a) ∗ R -∗ ∃ a, Ψ a ∗ P.
Proof.
  iIntros "[HP [HΨ HR]]".
  iDestruct "HΨ" as (x) "HΨ".
  iExists x.
  iSplitL "HΨ".
```

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

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

Implementation of the iSplit tactic

Tactics implemented by reflection as mere lemmas:

```
Lemma tac_sep_split Δ Δ1 Δ2 lr js Q1 Q2 :
  envs_split lr js Δ = Some (Δ1,Δ2) →
  (Δ1 ⊢ Q1) → (Δ2 ⊢ Q2) → Δ ⊢ Q1 ∗ Q2.
```

Implementation of the iSplit tactic

Tactics implemented by reflection as mere lemmas:

```
Lemma tac_sep_split Δ Δ1 Δ2 lr js Q1 Q2 :
  envs_split lr js Δ = Some (Δ1,Δ2) →
  (Δ1 ⊢ Q1) → (Δ2 ⊢ Q2) → Δ ⊢ Q1 ∗ Q2.
```

Context splitting implemented as a computable Coq function (= efficient)

Implementation of the iSplit tactic

Tactics implemented by reflection as mere lemmas:

```
Lemma tac_sep_split Δ Δ1 Δ2 lr js Q1 Q2 :
  envs_split lr js Δ = Some (Δ1,Δ2) →
  (Δ1 ⊢ Q1) → (Δ2 ⊢ Q2) → Δ ⊢ Q1 ∗ Q2.
```

Context splitting implemented as a computable Coq function (= efficient)

Ltac wrappers around the reflective tactic:

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

Implementation of the iSplit tactic

Tactics implemented by reflection as mere lemmas:

```
Lemma tac_sep_split Δ Δ1 Δ2 lr js Q1 Q2 :
  envs_split lr js Δ = Some (Δ1,Δ2) →
  (Δ1 ⊢ Q1) → (Δ2 ⊢ Q2) → Δ ⊢ Q1 ∗ Q2.
```

Context splitting implemented as a computable Coq function (= efficient)

Ltac wrappers around the reflective tactic:

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

Report sensible error to the user

Proving Hoare triples using IPM

Consider:

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

How to use IPM to manipulate the precondition?

Proving Hoare triples using IPM

Consider:

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

How to use IPM to manipulate the precondition?

Solution: define Hoare triple in terms of weakest preconditions

We let:

$$\{P\}\, e\, \{w.\, Q\} \triangleq \square(P \mathrel{-\!\!*} \text{wp}\, e\, \{w.\, Q\})$$

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

- all executions of e are safe
- if e terminates with value v, the final state satisfies $Q[v/w]$

Proving swap using symbolic execution

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

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

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

Proving swap using symbolic execution

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

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

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

Proving swap using symbolic execution

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

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

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

Proving swap using symbolic execution

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

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

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

Proving swap using symbolic execution

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

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

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

Proving swap using symbolic execution

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

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

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

Proving swap using symbolic execution

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

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

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

Proving swap using symbolic execution

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

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

```
No more subgoals.
```

Proving swap using symbolic execution

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

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

How does this work internally: rules for weakest preconditions

Let us just translate the Hoare rules naively:

$$\ell \mapsto v \quad -\!\!* \quad \mathsf{wp}\ \ !\ell\ \{w.\, w = v * \ell \mapsto v\}$$
$$\ell \mapsto \_ \quad -\!\!* \quad \mathsf{wp}\ \ (\ell := v')\ \{w.\, w = () * \ell \mapsto v'\}$$

How does this work internally: rules for weakest preconditions

Let us just translate the Hoare rules naively:

$$\ell \mapsto v \quad \twoheadrightarrow \quad \mathsf{wp}\ \ !\ell\ \{w.\, w = v * \ell \mapsto v\}$$
$$\ell \mapsto \_ \quad \twoheadrightarrow \quad \mathsf{wp}\ \ (\ell := v')\ \{w.\, w = () * \ell \mapsto v'\}$$

Problems: these rules are sound, but are not suitable for interactive proofs:

- To use these rules, the postcondition Q should be of a very specific shape
- Need to frame and weaken to use the rule

How does this work internally: rules for weakest preconditions

Let us just translate the Hoare rules naively:

$$\ell \mapsto v \quad \twoheadrightarrow \quad \mathsf{wp}\ \ !\ell\ \{w.\, w = v * \ell \mapsto v\}$$
$$\ell \mapsto \_ \quad \twoheadrightarrow \quad \mathsf{wp}\ \ (\ell := v')\ \{w.\, w = () * \ell \mapsto v'\}$$

Problems: these rules are sound, but are not suitable for interactive proofs:

- To use these rules, the postcondition Q should be of a very specific shape
- Need to frame and weaken to use the rule

Better approach: use 'backwards'/'predicate transformer' rules for "`wp_`" tactics:

$$\ell \mapsto v \quad * \quad (\ell \mapsto v \twoheadrightarrow \Phi\, v) \quad \twoheadrightarrow \quad \mathsf{wp}\ \ !\ell\ \{\Phi\}$$
$$\ell \mapsto \_ \quad * \quad (\ell \mapsto v' \twoheadrightarrow \Phi\, ()) \quad \twoheadrightarrow \quad \mathsf{wp}\ \ (\ell := v')\ \{\Phi\}$$

Resources that have to be given up

Resources that are given back

Using the rules for weakest preconditions

$$\frac{}{\ell_1 \mapsto v_1 * \ell_2 \mapsto v_2 \vdash \mathsf{wp}\,(\ell_1 := v_2; \ell_2 := v_1)\,\{\ell_1 \mapsto v_2 * \ell_2 \mapsto v_1\}}\ \text{WP-SEQ}$$

Using the rules for weakest preconditions

$$\dfrac{\dfrac{}{\ell_1 \mapsto v_1 * \ell_2 \mapsto v_2 \vdash \mathsf{wp}\,(\ell_1 := v_2)\,\{\mathsf{wp}\,(\ell_2 := v_1)\,\{\ell_1 \mapsto v_2 * \ell_2 \mapsto v_1\}\}}\ \text{WP-STORE}}{\ell_1 \mapsto v_1 * \ell_2 \mapsto v_2 \vdash \mathsf{wp}\,(\ell_1 := v_2; \ell_2 := v_1)\,\{\ell_1 \mapsto v_2 * \ell_2 \mapsto v_1\}}\ \text{WP-SEQ}$$

Using the rules for weakest preconditions

$$\dfrac{\dfrac{\dfrac{}{\ell_1 \mapsto v_1 * \ell_2 \mapsto v_2 \vdash \ell_1 \mapsto \_ * (\ell_1 \mapsto v_2 \mathrel{-\!\!*} \mathsf{wp}\,(\ell_2 := v_1)\,\{\ell_1 \mapsto v_2 * \ell_2 \mapsto v_1\})}\ *\text{-MONO}}{\ell_1 \mapsto v_1 * \ell_2 \mapsto v_2 \vdash \mathsf{wp}\,(\ell_1 := v_2)\,\{\mathsf{wp}\,(\ell_2 := v_1)\,\{\ell_1 \mapsto v_2 * \ell_2 \mapsto v_1\}\}}\ \text{WP-STORE}}{\ell_1 \mapsto v_1 * \ell_2 \mapsto v_2 \vdash \mathsf{wp}\,(\ell_1 := v_2; \ell_2 := v_1)\,\{\ell_1 \mapsto v_2 * \ell_2 \mapsto v_1\}}\ \text{WP-SEQ}$$

Using the rules for weakest preconditions

$$\dfrac{\dfrac{\dfrac{}{\ell_2 \mapsto v_2 \vdash \ell_1 \mapsto v_2 \mathrel{-\!\!*} \mathsf{wp}\,(\ell_2 := v_1)\,\{\ell_1 \mapsto v_2 * \ell_2 \mapsto v_1\}}\,\mathrel{-\!\!*}\text{-INTRO}}{\ell_1 \mapsto v_1 * \ell_2 \mapsto v_2 \vdash \ell_1 \mapsto \_ * (\ell_1 \mapsto v_2 \mathrel{-\!\!*} \mathsf{wp}\,(\ell_2 := v_1)\,\{\ell_1 \mapsto v_2 * \ell_2 \mapsto v_1\})}\,*\text{-MONO}}{\ell_1 \mapsto v_1 * \ell_2 \mapsto v_2 \vdash \mathsf{wp}\,(\ell_1 := v_2)\,\{\mathsf{wp}\,(\ell_2 := v_1)\,\{\ell_1 \mapsto v_2 * \ell_2 \mapsto v_1\}\}}\,\text{WP-STORE}$$

$$\dfrac{\ell_1 \mapsto v_1 * \ell_2 \mapsto v_2 \vdash \mathsf{wp}\,(\ell_1 := v_2; \ell_2 := v_1)\,\{\ell_1 \mapsto v_2 * \ell_2 \mapsto v_1\}}{\ell_1 \mapsto v_1 * \ell_2 \mapsto v_2 \vdash \mathsf{wp}\,(\ell_1 := v_2)\,\{\mathsf{wp}\,(\ell_2 := v_1)\,\{\ell_1 \mapsto v_2 * \ell_2 \mapsto v_1\}\}}\,\text{WP-SEQ}$$

Using the rules for weakest preconditions

$$\dfrac{\dfrac{\dfrac{}{\ell_2 \mapsto v_2 \vdash \ell_1 \mapsto v_2 \mathrel{-\!\!*} \mathsf{wp}\,(\ell_2 := v_1)\,\{\ell_1 \mapsto v_2 * \ell_2 \mapsto v_1\}}\,\mathrel{-\!\!*}\text{-INTRO}}{\ell_1 \mapsto v_1 * \ell_2 \mapsto v_2 \vdash \ell_1 \mapsto \_ * (\ell_1 \mapsto v_2 \mathrel{-\!\!*} \mathsf{wp}\,(\ell_2 := v_1)\,\{\ell_1 \mapsto v_2 * \ell_2 \mapsto v_1\})}\,*\text{-MONO}}{\ell_1 \mapsto v_1 * \ell_2 \mapsto v_2 \vdash \mathsf{wp}\,(\ell_1 := v_2)\,\{\mathsf{wp}\,(\ell_2 := v_1)\,\{\ell_1 \mapsto v_2 * \ell_2 \mapsto v_1\}\}}\,\text{WP-STORE}$$

$$\dfrac{\ell_1 \mapsto v_1 * \ell_2 \mapsto v_2 \vdash \mathsf{wp}\,(\ell_1 := v_2; \ell_2 := v_1)\,\{\ell_1 \mapsto v_2 * \ell_2 \mapsto v_1\}}{\ell_1 \mapsto v_1 * \ell_2 \mapsto v_2 \vdash \mathsf{wp}\,(\ell_1 := v_2)\,\{\mathsf{wp}\,(\ell_2 := v_1)\,\{\ell_1 \mapsto v_2 * \ell_2 \mapsto v_1\}\}}\,\text{WP-SEQ}$$

Using the rules for weakest preconditions

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\ell_1 \mapsto v_2 * \ell_2 \mapsto v_2 \vdash \mathsf{wp}\,(\ell_2 := v_1)\,\{\ell_1 \mapsto v_2 * \ell_2 \mapsto v_1\}}\,\text{WP-STORE}}{\ell_2 \mapsto v_2 \vdash \ell_1 \mapsto v_2 \mathrel{-\!\!*} \mathsf{wp}\,(\ell_2 := v_1)\,\{\ell_1 \mapsto v_2 * \ell_2 \mapsto v_1\}}\,\mathrel{-\!\!*}\text{-INTRO}}{\ell_1 \mapsto v_1 * \ell_2 \mapsto v_2 \vdash \ell_1 \mapsto \_ * (\ell_1 \mapsto v_2 \mathrel{-\!\!*} \mathsf{wp}\,(\ell_2 := v_1)\,\{\ell_1 \mapsto v_2 * \ell_2 \mapsto v_1\})}\,*\text{-MONO}}{\ell_1 \mapsto v_1 * \ell_2 \mapsto v_2 \vdash \mathsf{wp}\,(\ell_1 := v_2)\,\{\mathsf{wp}\,(\ell_2 := v_1)\,\{\ell_1 \mapsto v_2 * \ell_2 \mapsto v_1\}\}}\,\text{WP-STORE}}{\ell_1 \mapsto v_1 * \ell_2 \mapsto v_2 \vdash \mathsf{wp}\,(\ell_1 := v_2; \ell_2 := v_1)\,\{\ell_1 \mapsto v_2 * \ell_2 \mapsto v_1\}}\,\text{WP-SEQ}$$

Using the rules for weakest preconditions

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\ell_1 \mapsto v_2 * \ell_2 \mapsto v_2 \vdash \ell_2 \mapsto \_ * (\ell_2 \mapsto v_1 \mathrel{-\!\!*} (\ell_1 \mapsto v_2 * \ell_2 \mapsto v_1))}\,*\text{-MONO}}{\ell_1 \mapsto v_2 * \ell_2 \mapsto v_2 \vdash \mathsf{wp}\,(\ell_2 := v_1)\,\{\ell_1 \mapsto v_2 * \ell_2 \mapsto v_1\}}\,\text{WP-STORE}}{\ell_2 \mapsto v_2 \vdash \ell_1 \mapsto v_2 \mathrel{-\!\!*} \mathsf{wp}\,(\ell_2 := v_1)\,\{\ell_1 \mapsto v_2 * \ell_2 \mapsto v_1\}}\,\mathrel{-\!\!*}\text{-INTRO}}{\ell_1 \mapsto v_1 * \ell_2 \mapsto v_2 \vdash \ell_1 \mapsto \_ * (\ell_1 \mapsto v_2 \mathrel{-\!\!*} \mathsf{wp}\,(\ell_2 := v_1)\,\{\ell_1 \mapsto v_2 * \ell_2 \mapsto v_1\})}\,*\text{-MONO}}{\ell_1 \mapsto v_1 * \ell_2 \mapsto v_2 \vdash \mathsf{wp}\,(\ell_1 := v_2)\,\{\mathsf{wp}\,(\ell_2 := v_1)\,\{\ell_1 \mapsto v_2 * \ell_2 \mapsto v_1\}\}}\,\text{WP-STORE}}{\ell_1 \mapsto v_1 * \ell_2 \mapsto v_2 \vdash \mathsf{wp}\,(\ell_1 := v_2; \ell_2 := v_1)\,\{\ell_1 \mapsto v_2 * \ell_2 \mapsto v_1\}}\,\text{WP-SEQ}$$

Using the rules for weakest preconditions

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\ell_1 \mapsto v_2 * \ell_2 \mapsto v_2 \vdash \ell_2 \mapsto \_ * (\ell_2 \mapsto v_1 \mathrel{-\!\!*} (\ell_1 \mapsto v_2 * \ell_2 \mapsto v_1))}\,*\text{-MONO}}{\ell_1 \mapsto v_2 * \ell_2 \mapsto v_2 \vdash \mathsf{wp}\,(\ell_2 := v_1)\,\{\ell_1 \mapsto v_2 * \ell_2 \mapsto v_1\}}\,\text{WP-STORE}}{\ell_2 \mapsto v_2 \vdash \ell_1 \mapsto v_2 \mathrel{-\!\!*} \mathsf{wp}\,(\ell_2 := v_1)\,\{\ell_1 \mapsto v_2 * \ell_2 \mapsto v_1\}}\,\mathrel{-\!\!*}\text{-INTRO}}{\ell_1 \mapsto v_1 * \ell_2 \mapsto v_2 \vdash \ell_1 \mapsto \_ * (\ell_1 \mapsto v_2 \mathrel{-\!\!*} \mathsf{wp}\,(\ell_2 := v_1)\,\{\ell_1 \mapsto v_2 * \ell_2 \mapsto v_1\})}\,*\text{-MONO}}{\ell_1 \mapsto v_1 * \ell_2 \mapsto v_2 \vdash \mathsf{wp}\,(\ell_1 := v_2)\,\{\mathsf{wp}\,(\ell_2 := v_1)\,\{\ell_1 \mapsto v_2 * \ell_2 \mapsto v_1\}\}}\,\text{WP-STORE}}{\ell_1 \mapsto v_1 * \ell_2 \mapsto v_2 \vdash \mathsf{wp}\,(\ell_1 := v_2; \ell_2 := v_1)\,\{\ell_1 \mapsto v_2 * \ell_2 \mapsto v_1\}}\,\text{WP-SEQ}$$

Using the rules for weakest preconditions

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\ell_1 \mapsto v_2 \vdash \ell_2 \mapsto v_1 \mathrel{-\!\!*} (\ell_1 \mapsto v_2 * \ell_2 \mapsto v_1)}\,\mathrel{-\!\!*}\text{-INTRO}}{\ell_1 \mapsto v_2 * \ell_2 \mapsto v_2 \vdash \ell_2 \mapsto \_ * (\ell_2 \mapsto v_1 \mathrel{-\!\!*} (\ell_1 \mapsto v_2 * \ell_2 \mapsto v_1))}\,*\text{-MONO}}{\ell_1 \mapsto v_2 * \ell_2 \mapsto v_2 \vdash \mathsf{wp}\,(\ell_2 := v_1)\,\{\ell_1 \mapsto v_2 * \ell_2 \mapsto v_1\}}\,\text{WP-STORE}}{\ell_2 \mapsto v_2 \vdash \ell_1 \mapsto v_2 \mathrel{-\!\!*} \mathsf{wp}\,(\ell_2 := v_1)\,\{\ell_1 \mapsto v_2 * \ell_2 \mapsto v_1\}}\,\mathrel{-\!\!*}\text{-INTRO}}{\ell_1 \mapsto v_1 * \ell_2 \mapsto v_2 \vdash \ell_1 \mapsto \_ * (\ell_1 \mapsto v_2 \mathrel{-\!\!*} \mathsf{wp}\,(\ell_2 := v_1)\,\{\ell_1 \mapsto v_2 * \ell_2 \mapsto v_1\})}\,*\text{-MONO}}{\ell_1 \mapsto v_1 * \ell_2 \mapsto v_2 \vdash \mathsf{wp}\,(\ell_1 := v_2)\,\{\mathsf{wp}\,(\ell_2 := v_1)\,\{\ell_1 \mapsto v_2 * \ell_2 \mapsto v_1\}\}}\,\text{WP-STORE}}{\ell_1 \mapsto v_1 * \ell_2 \mapsto v_2 \vdash \mathsf{wp}\,(\ell_1 := v_2; \ell_2 := v_1)\,\{\ell_1 \mapsto v_2 * \ell_2 \mapsto v_1\}}\,\text{WP-SEQ}$$

Using the rules for weakest preconditions

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\ell_1 \mapsto v_2 \vdash \ell_2 \mapsto v_1 \mathrel{-\!\!*} (\ell_1 \mapsto v_2 * \ell_2 \mapsto v_1)}\ \mathrel{-\!\!*}\text{-INTRO}}{\ell_1 \mapsto v_2 * \ell_2 \mapsto v_2 \vdash \ell_2 \mapsto \_ * (\ell_2 \mapsto v_1 \mathrel{-\!\!*} (\ell_1 \mapsto v_2 * \ell_2 \mapsto v_1))}\ *\text{-MONO}}{\ell_1 \mapsto v_2 * \ell_2 \mapsto v_2 \vdash \mathsf{wp}\,(\ell_2 := v_1)\,\{\ell_1 \mapsto v_2 * \ell_2 \mapsto v_1\}}\ \text{WP-STORE}}{\ell_2 \mapsto v_2 \vdash \ell_1 \mapsto v_2 \mathrel{-\!\!*} \mathsf{wp}\,(\ell_2 := v_1)\,\{\ell_1 \mapsto v_2 * \ell_2 \mapsto v_1\}}\ \mathrel{-\!\!*}\text{-INTRO}}{\ell_1 \mapsto v_1 * \ell_2 \mapsto v_2 \vdash \ell_1 \mapsto \_ * (\ell_1 \mapsto v_2 \mathrel{-\!\!*} \mathsf{wp}\,(\ell_2 := v_1)\,\{\ell_1 \mapsto v_2 * \ell_2 \mapsto v_1\})}\ *\text{-MONO}}{\ell_1 \mapsto v_1 * \ell_2 \mapsto v_2 \vdash \mathsf{wp}\,(\ell_1 := v_2)\,\{\mathsf{wp}\,(\ell_2 := v_1)\,\{\ell_1 \mapsto v_2 * \ell_2 \mapsto v_1\}\}}\ \text{WP-STORE}}{\ell_1 \mapsto v_1 * \ell_2 \mapsto v_2 \vdash \mathsf{wp}\,(\ell_1 := v_2; \ell_2 := v_1)\,\{\ell_1 \mapsto v_2 * \ell_2 \mapsto v_1\}}\ \text{WP-SEQ}$$

Using the rules for weakest preconditions

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\ell_1 \mapsto v_2 * \ell_2 \mapsto v_1 \vdash \ell_1 \mapsto v_2 * \ell_2 \mapsto v_1}}{\ell_1 \mapsto v_2 \vdash \ell_2 \mapsto v_1 \mathrel{-\!\!*} (\ell_1 \mapsto v_2 * \ell_2 \mapsto v_1)}\ \mathrel{-\!\!*}\text{-INTRO}}{\ell_1 \mapsto v_2 * \ell_2 \mapsto v_2 \vdash \ell_2 \mapsto \_ * (\ell_2 \mapsto v_1 \mathrel{-\!\!*} (\ell_1 \mapsto v_2 * \ell_2 \mapsto v_1))}\ *\text{-MONO}}{\ell_1 \mapsto v_2 * \ell_2 \mapsto v_2 \vdash \mathsf{wp}\,(\ell_2 := v_1)\,\{\ell_1 \mapsto v_2 * \ell_2 \mapsto v_1\}}\ \text{WP-STORE}}{\ell_2 \mapsto v_2 \vdash \ell_1 \mapsto v_2 \mathrel{-\!\!*} \mathsf{wp}\,(\ell_2 := v_1)\,\{\ell_1 \mapsto v_2 * \ell_2 \mapsto v_1\}}\ \mathrel{-\!\!*}\text{-INTRO}}{\ell_1 \mapsto v_1 * \ell_2 \mapsto v_2 \vdash \ell_1 \mapsto \_ * (\ell_1 \mapsto v_2 \mathrel{-\!\!*} \mathsf{wp}\,(\ell_2 := v_1)\,\{\ell_1 \mapsto v_2 * \ell_2 \mapsto v_1\})}\ *\text{-MONO}}{\ell_1 \mapsto v_1 * \ell_2 \mapsto v_2 \vdash \mathsf{wp}\,(\ell_1 := v_2)\,\{\mathsf{wp}\,(\ell_2 := v_1)\,\{\ell_1 \mapsto v_2 * \ell_2 \mapsto v_1\}\}}\ \text{WP-STORE}}{\ell_1 \mapsto v_1 * \ell_2 \mapsto v_2 \vdash \mathsf{wp}\,(\ell_1 := v_2; \ell_2 := v_1)\,\{\ell_1 \mapsto v_2 * \ell_2 \mapsto v_1\}}\ \text{WP-SEQ}$$

Using the rules for weakest preconditions

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\ell_1 \mapsto v_2 * \ell_2 \mapsto v_1 \vdash \ell_1 \mapsto v_2 * \ell_2 \mapsto v_1}}{\ell_1 \mapsto v_2 \vdash \ell_2 \mapsto v_1 \mathrel{-\!\!*} (\ell_1 \mapsto v_2 * \ell_2 \mapsto v_1)}\ \mathrel{-\!\!*}\text{-INTRO}}{\ell_1 \mapsto v_2 * \ell_2 \mapsto v_2 \vdash \ell_2 \mapsto \_ * (\ell_2 \mapsto v_1 \mathrel{-\!\!*} (\ell_1 \mapsto v_2 * \ell_2 \mapsto v_1))}\ *\text{-MONO}}{\ell_1 \mapsto v_2 * \ell_2 \mapsto v_2 \vdash \mathsf{wp}\,(\ell_2 := v_1)\,\{\ell_1 \mapsto v_2 * \ell_2 \mapsto v_1\}}\ \text{WP-STORE}}{\ell_2 \mapsto v_2 \vdash \ell_1 \mapsto v_2 \mathrel{-\!\!*} \mathsf{wp}\,(\ell_2 := v_1)\,\{\ell_1 \mapsto v_2 * \ell_2 \mapsto v_1\}}\ \mathrel{-\!\!*}\text{-INTRO}}{\ell_1 \mapsto v_1 * \ell_2 \mapsto v_2 \vdash \ell_1 \mapsto \_ * (\ell_1 \mapsto v_2 \mathrel{-\!\!*} \mathsf{wp}\,(\ell_2 := v_1)\,\{\ell_1 \mapsto v_2 * \ell_2 \mapsto v_1\})}\ *\text{-MONO}}{\ell_1 \mapsto v_1 * \ell_2 \mapsto v_2 \vdash \mathsf{wp}\,(\ell_1 := v_2)\,\{\mathsf{wp}\,(\ell_2 := v_1)\,\{\ell_1 \mapsto v_2 * \ell_2 \mapsto v_1\}\}}\ \text{WP-STORE}}{\ell_1 \mapsto v_1 * \ell_2 \mapsto v_2 \vdash \mathsf{wp}\,(\ell_1 := v_2; \ell_2 := v_1)\,\{\ell_1 \mapsto v_2 * \ell_2 \mapsto v_1\}}\ \text{WP-SEQ}$$

Remember: the following pattern is very useful to build separation logic tactics:

"old resources" $\mathrel{-\!\!*}$ ("new resources" $\mathrel{-\!\!*}$ "new goal") $\mathrel{-\!\!*}$ "goal"

Concurrent example

The racy example from the previous (Derek's) talk:

$$
\begin{array}{l}
\{\mathsf{True}\} \\
\texttt{let}\, x = \texttt{ref}(0)\, \texttt{in} \\
\texttt{fetchandadd}(x, 2) \parallel \texttt{fetchandadd}(x, 2) \\
!\, x \\
\{w.\, w = 4\}
\end{array}
$$

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

Concurrent example

The racy example from the previous (Derek's) talk:

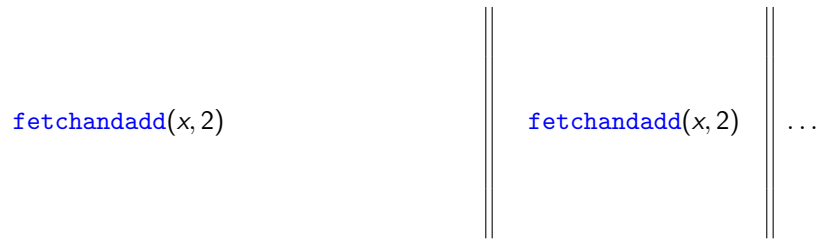
$$
\begin{array}{l}
\{\mathsf{True}\} \\
\texttt{let}\, x = \texttt{ref}(0)\, \texttt{in} \\
\texttt{fetchandadd}(x, 2) \parallel \texttt{fetchandadd}(x, 2) \\
!\, x \\
\{w.\, w = 4\}
\end{array}
$$

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

$$
\begin{array}{l}
\texttt{fetchandadd}(x, y) \triangleq \texttt{let}\, n = !\, x\, \texttt{in} \\
\qquad\qquad\qquad\qquad\quad \texttt{if}\, \texttt{CAS}(x, n, n + y)\, \texttt{then}\, n \\
\qquad\qquad\qquad\qquad\quad \texttt{else}\, \texttt{fetchandadd}(x, y)
\end{array}
$$

Recap of the proof

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

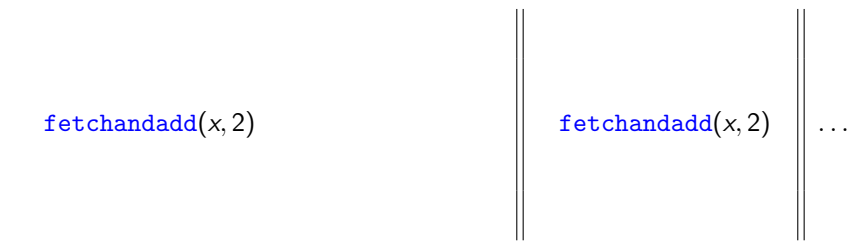


$!\,x$

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

Recap of the proof

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



$!\,x$

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

Recap of the proof

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

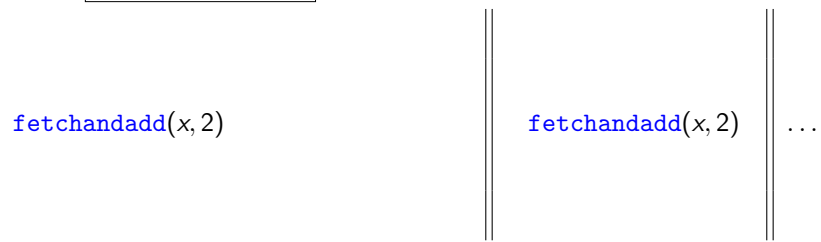
$\texttt{fetchandadd}(x, 2)$ $\quad\|\quad$ $\texttt{fetchandadd}(x, 2)$ $\quad\|\quad \ldots$

$!\, x$

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

Recap of the proof

$\{\mathsf{True}\}$
$\texttt{let}\, x = \texttt{ref}(0)\, \texttt{in}$
$\{x \mapsto 0\}$
$\left\{x \mapsto 0 * \gamma \hookrightarrow_\bullet 0 * \gamma \overset{1}{\hookrightarrow}_\circ 0\right\}$
allocate $\boxed{\exists n.\, x \mapsto n * \gamma \hookrightarrow_\bullet n}$



$!\, x$

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

Recap of the proof

$\{\mathsf{True}\}$

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

$\{x \mapsto 0\}$

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

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

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

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

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

$!\, x$

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

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

Recap of the proof

$\{\mathsf{True}\}$

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

$\{x \mapsto 0\}$

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

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

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

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

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

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

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

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

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

$\ldots$

$!\, x$

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

Recap of the proof

$\{\mathsf{True}\}$

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

$\{x \mapsto 0\}$

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

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

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

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

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

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

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

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

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

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

$\ldots$

$!\, x$

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

Recap of the proof

$\{\mathsf{True}\}$

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

$\{x \mapsto 0\}$

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

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

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

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

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

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

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

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

$\{\dots\}$

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

$\{\dots\}$

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

$\dots$

$!\, x$

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

Recap of the proof

$\{\mathsf{True}\}$

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

$\{x \mapsto 0\}$

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

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

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

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

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

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

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

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

$!\, x$

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

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

$\{\dots\}$

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

$\{\dots\}$

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

$\dots$

Recap of the proof

$\{\mathsf{True}\}$
$\texttt{let}\, x = \texttt{ref}(0)\, \texttt{in}$
$\{x \mapsto 0\}$
$\left\{x \mapsto 0 * \gamma \hookrightarrow_\bullet 0 * \gamma \overset{1}{\hookrightarrow}_\circ 0\right\}$
allocate $\boxed{\exists n.\, x \mapsto n * \gamma \hookrightarrow_\bullet n}$

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

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

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

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

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

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

$!\, x$

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

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

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

$\{\ldots\}$

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

$\{\ldots\}$

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

$\ldots$

The proof in IPM

Making IPM tactics modular using type classes

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

- turn `H : P * Q` into `H1 : P` and `H2 : Q`
- turn `H : ▷(P * Q)` into `H2 : ▷ P` and `H2 : ▷ Q`
- turn `H : own γ (◯!{q} (n1 + n2))` into
  `H : own γ (◯!{q/2} n1)` and `H : own γ (◯!{q/2} n2)`

Making IPM tactics modular using type classes

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

- turn `H : P * Q` into `H1 : P` and `H2 : Q`
- turn `H : ▷(P * Q)` into `H2 : ▷ P` and `H2 : ▷ Q`
- turn `H : own γ (◯!{q} (n1 + n2))` into
  `H : own γ (◯!{q/2} n1)` and `H : own γ (◯!{q/2} n2)`

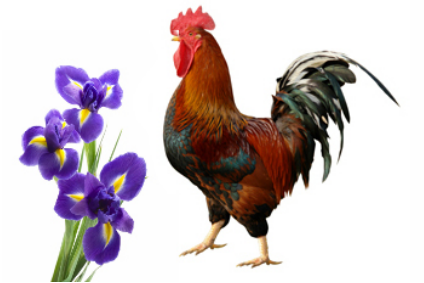
We use type classes to achieve that:

```
Class IntoAnd (p : bool) (P Q1 Q2 : uPred M) :=
  into_and : P ⊢ if p then Q1 ∧ Q2 else Q1 ∗ Q2.
Instance into_and_sep p P Q : IntoAnd p (P ∗ Q) P Q.
Instance into_and_and P Q : IntoAnd true (P ∧ Q) P Q.
Instance into_and_later p P Q1 Q2 : IntoAnd p P Q1 Q2 → IntoAnd p (▷ P) (▷ Q1) (▷ Q2).

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

IPM in summary

- Propositions are shallowly embedded
- Contexts are deeply embedded
- Context manipulation is done via **computational reflection**
- IPM tactics are just Coq lemmas
- **Type classes** are used to make the tactics more general
- **Ltac** is used to provide an end-user syntax and error reporting
- Backward **weakest precondition** rules are well-suited for interactive proofs



IPM in summary

- Propositions are shallowly embedded
- Contexts are deeply embedded
- Context manipulation is done via **computational reflection**
- IPM tactics are just Coq lemmas
- **Type classes** are used to make the tactics more general
- **Ltac** is used to provide an end-user syntax and error reporting
- Backward **weakest precondition** rules are well-suited for interactive proofs

These ideas are hopefully applicable to other object logics

Future work

- Make IPM independent of the Iris logic
- Support for logical atomicity in IPM
- Interactive proofs for program refinements in IPM

Thank you, and download Iris/IPM at `http://iris-project.org/`