Mechanized proofs in higher-order separation logic

Robbert Krebbers

Delft University of Technology, The Netherlands

February 5, 2019 © Vrije Universiteit, Amsterdam, The Netherlands
Tactic-style proofs (as in LCF/Coq/HOL/etc.) have shown to be effective in large-scale proof developments (CompCert, Four color, Feit-Thompson, Kepler, ...)
Basic example of tactic-style proofs in Coq

Lemma test {A} (P Q : Prop) (Ψ : A → Prop) :
P ∧ (∃ a, Ψ a) ∧ Q → Q ∧ ∃ a, P ∧ Ψ a.

Proof.
intros [H1 [H2 H3]].
destruct H2 as [x H2].
split.
- assumption.
- exists x.
  auto.
Qed.
Basic example of tactic-style proofs in Coq

Lemma test {A} (P Q : Prop) (Ψ : A → Prop) :
  P ∧ (∃ a, Ψ a) ∧ Q → Q ∧ ∃ a, P ∧ Ψ a.

Proof.
  intros [H1 [H2 H3]].
  destruct H2 as [x H2].
  split.
  - assumption.
  - exists x.
    auto.

Qed.
Basic example of tactic-style proofs in Coq

Lemma test \{A\} (P Q : Prop) (Ψ : A → Prop) :
    P ∧ (∃ a, Ψ a) ∧ Q → Q ∧ ∃ a, P ∧ Ψ a.
Proof.
    intros [H1 [H2 H3]].
destruct H2 as [x H2].
split.
    - assumption.
    - exists x.
        auto.
Qed.

Context

Goal

1 subgoal
A : Type
P, Q : Prop
Ψ : A → Prop
H1 : P
H2 : ∃ a : A, Ψ a
H3 : Q

Q ∧ (∃ a : A, P ∧ Ψ a)
Basic example of tactic-style proofs in Coq

Lemma test \{A\} (P Q : Prop) (Ψ : A → Prop) :
    P ∧ (∃ a, Ψ a) ∧ Q → Q ∧ ∃ a, P ∧ Ψ a.

Proof.
    intros [H1 [H2 H3]].
    destruct H2 as [x H2].
    split.
    - assumption.
    - exists x.
        auto.
Qed.

1 subgoal
A : Type
P, Q : Prop
Ψ : A → Prop
H1 : P
H2 : ∃ a : A, Ψ a
H3 : Q
___________________________(1/1)
Q ∧ (∃ a : A, P ∧ Ψ a)
Lemma test {A} (P Q : Prop) (Ψ : A → Prop) :
P ∧ (∃ a, Ψ a) ∧ Q → Q ∧ ∃ a, P ∧ Ψ a.

Proof.
intros [H1 [H2 H3]].
destruct H2 as [x H2].
split.
- assumption.
- exists x.
  auto.
Qed.
**Lemma** test \{A\} (P Q : Prop) (Ψ : A → Prop) :
\[ P \land (\exists a, Ψ a) \land Q → Q \land \exists a, P \land Ψ a. \]

**Proof.**
intros [H1 [H2 H3]].
destruct H2 as [x H2].

split.
- assumption.
- exists x.
  auto.

Qed.
**Basic example of tactic-style proofs in Coq**

**Lemma** test \( \{A\} (P \ Q : \text{Prop}) (\Psi : A \rightarrow \text{Prop}) : \)
\[ P \land (\exists \ a, \Psi \ a) \land Q \rightarrow Q \land \exists \ a, \ P \land \Psi \ a. \]

**Proof.**

intros [H1 [H2 H3]].
destruct H2 as [x H2].
split.
- assumption.
- exists x.
  auto.

Qed.

---

2 subgoals

- \( A : \text{Type} \)
- \( P, \ Q : \text{Prop} \)
- \( \Psi : A \rightarrow \text{Prop} \)
- \( H1 : P \)
- \( x : A \)
- \( H2 : \Psi \ x \)
- \( H3 : Q \)

\[ \exists \ a : A, \ P \land \Psi \ a \]
Lemma test \( \{A\} \) (\( P \) \( Q : \) Prop) (\( \Psi : A \rightarrow \) Prop) : 
\( P \land (\exists a, \Psi a) \land Q \rightarrow Q \land \exists a, P \land \Psi a. \)

Proof.

intros [H1 [H2 H3]].
destruct H2 as [x H2].
split.
- assumption.
- exists x.
  auto.

Qed.
Basic example of tactic-style proofs in Coq

```
Lemma test \{A\} (P Q : Prop) (Ψ : A → Prop) :
  P ∧ (∃ a, Ψ a) ∧ Q → Q ∧ ∃ a, P ∧ Ψ a.
Proof.
  intros [H1 [H2 H3]].
  by firstorder (* automate this *).
Qed.
```

Scales in practice

- High-level tactics for arithmetic, Prolog-style search, algebra, …
- Compact syntax for combining tactics (ssreflect)
- Tactic programming (using ML, Ltac, …)
Tactic-style proofs for other logics, like *separation logic*
Separation logic [O’Hearn, Reynolds, Yang; CSL’01]

Propositions $P$, $Q$ denote ownership of resources

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

Basic example:

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

the $\ast$ ensures that $x$ and $y$ are different memory locations
Why is separation logic useful?

Separation logic is very useful:

▶ It provides a high level of modularity
▶ It scales to fancy PL features like concurrency

Just in Coq, there is an ever growing collection of separation logics:

▶ Bedrock
▶ CFML
▶ Charge!
▶ CHL
▶ FCSL
▶ Iris
▶ VST
▶ …
Problem:
Cannot reuse the tactics/context of the proof assistant when reasoning in an embedded logic like separation logic
Goal of this talk

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)
In this paper, we introduce a so-called proof mode that extends the Coq proof assistant with in-place and non-in-place named proof contexts for the object logic. We show that thanks to these features, we can implement high-level tactics for introduction and elimination of the connectives of the object logic, and thereby make reasoning in the embedded logic as seamless as reasoning in the meta-logic of the proof assistant. We apply our method to Iris, a state of the art interactive proof assistant. We show that our method is very generic, and can be applied to a variety of different embedded logics, we apply it to a specific logic, use the proof assistant to reason in the embedded logic.

For many separation logics

MoSeL: A General, Extensible Modal Framework for Interactive Proofs in Separation Logic

ROBBERT KREBBERS, Delft University of Technology, The Netherlands
JACQUES-HENRI JOURDAN, LRI, Univ. Paris-Sud, CNRS, Université Paris-Saclay, France
RALF JUNG, MPI-SWS, Germany
JOSEPH TASSAROTTI, Carnegie Mellon University, USA
JAN-Olivier KAISER, MPI-SWS, Germany
AMIN TIMANY, imec-DistriNet, KU Leuven, Belgium
ARTHUR CHARGUÈRAUD, Inria & Université de Strasbourg, CNRS, ICube, France
DEREK DREYER, MPI-SWS, Germany

A number of tools have been developed for carrying out separation-logic proofs mechanically using an interactive proof assistant. One of the most advanced such tools is the Iris Proof Mode (IPM) for Coq, which offers a rich set of tactics for making separation-logic proofs look and feel like ordinary Coq proofs. However, IPM is tied to a particular separation logic (namely, Iris), thus limiting its applicability.

In this paper, we propose MoSeL, a general and extensible Coq framework that brings the benefits of IPM to a much larger class of separation logics. Unlike IPM, MoSeL is applicable to both affine and linear separation logics (and combinations thereof), and provides generic tactics that can be easily extended to account for the bespoke connectives of the logics with which it is instantiated. To demonstrate the effectiveness of MoSeL, we have instantiated it to provide effective tactical support for interactive and semi-automated proofs in six very
Iris [Jung, Krebbers et al.; POPL'15, ICFP'16, ESOP'17, JFP'18]

A general, language-independent, framework for modeling your own
domain specific higher-order separation logics
A general, language-independent, framework for modeling your own domain specific higher-order separation logics

- **General**: unifies the reasoning principles in many other logics
A general, language-independent, framework for modeling your own domain specific higher-order separation logics

- **General**: unifies the reasoning principles in many other logics
- **Language-independent**: parameterized by the language
Iris [Jung, Krebbers et al.; POPL’15, ICFP’16, ESOP’17, JFP’18]

A general, language-independent, framework for modeling your own domain specific higher-order separation logics

- **General**: unifies the reasoning principles in many other logics
- **Language-independent**: parameterized by the language
- **Modeling logics**: can be used to model domain specific logics
  - iGPS for weak memory [ECOOP’17]
  - RustBelt’s lifetime logic [POPL’18]
  - ReLoC for program refinements [LICS’18]
  - Iron for resource management [POPL’19]
Lemma test \{A\} (P Q : iProp) (Ψ : A → iProp) :
  P ∗ (∃ a, Ψ a) ∗ Q →∗ Q ∗ ∃ a, P ∗ Ψ a.

Proof.
  iIntros "[H1 [H2 H3]]".
  iDestruct "H2" as (x) "H2".
  iSplitL "H3".
  - iAssumption.
  - iExists x.
    iFrame.

Qed.
Lemma test \{A\} (P \land Q : iProp) (Ψ : A \rightarrow iProp) : 
P * (∃ a, Ψ a) * Q \rightarrow Q * ∃ a, P * Ψ a.

Proof.

Lemma in the Iris logic

iIntros "[H1 [H2 H3]]".
iDestruct "H2" as (x) "H2".
iSplitL "H3".
- iAssumption.
- iExists x.
iFrame.
Qed.
Lemma test \{A\} (P Q : iProp) (Ψ : A \rightarrow iProp) :
\[ P \ast (\exists a, \Psi a) \ast Q \rightarrow Q \ast \exists a, P \ast \Psi a. \]

Proof.
- iIntros "[H1 [H2 H3]]".
- iDestruct "H2" as (x) "H2".
- iSplitL "H3".
- iAssumption.
- iExists x.
  iFrame.
Qed.

1 subgoal
A : Type
P, Q : iProp
Ψ : A \rightarrow iProp
x : A

\-----------------------------(1/1)
"H1" : P
"H2" : Ψ x
"H3" : Q

\-----------------------------*
Q \ast (\exists a : A, P \ast Ψ a)
Lemma test \{A\} (P Q : iProp) (Ψ : A → iProp) :
  P * (∃ a, Ψ a) * Q → Q * ∃ a, P * Ψ 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
Ψ : A → iProp
x : A

______________________________(1/1)
"H1" : P
"H2" : Ψ x
"H3" : Q

______________________________
Q * (∃ a : A, P * Ψ a)

* means: resources should be split
Lemma test \{A\} (P Q : iProp) (Ψ : A → iProp) :
  P * (∃ a, Ψ a) * Q → Q * ∃ a, P * Ψ 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
Ψ : A → iProp
x : A
_______________________________(1/1)
"H1" : P
"H2" : Ψ x
"H3" : Q
______________________________*
Q * (∃ a : A, P * Ψ a)

* means: resources should be split

The hypotheses for the left conjunct
Lemma test \{A\} (P Q : iProp) (Ψ : A → iProp):
  P * (∃ a, Ψ a) * Q → Q * ∃ a, P * Ψ 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
Ψ : A → iProp
x : A
__________________________ (1/2)
"H3" : Q
__________________________ (2/2)

Q
Ψ x
__________________________ (2/2)
∃ a : A, P * Ψ a

The hypotheses for the left conjunct
Lemma test \{A\} (P Q : iProp) (Ψ : A → iProp) :

\[ P \ast (\exists a, \Psi a) \ast Q \not\ast Q \not\ast \exists a, P \ast \Psi a. \]

Proof.

iIntros "[H1 [H2 H3]]".

iDestruct "H2" as (x) "H2".

iSplitL "H3".

- iAssumption.

- iExists x.

  iFrame.

Qed.
Lemma test \{A\} (P Q : iProp) (Ψ : A → iProp) :
  P * (\exists a, Ψ a) * Q →* Q * \exists a, P * Ψ a.

Proof.
  iIntros "[H1 [H2 H3]]".
  by iFrame.

Qed.

We can also solve this lemma automatically
Scaling up

Iris Proof Mode scales and is used for any project involving Iris today because:

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

Iris Proof Mode scales and is used for any project involving Iris today because:

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

- **Support for advanced features of separation logic**
  Higher-order quantification, invariants, ghost state, later
  - modality, . . .
Scaling up

Iris Proof Mode scales and is used for any project involving Iris today because:

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

- **Support for advanced features of separation logic**
  Higher-order quantification, invariants, ghost state, later
  ⊢ modality, . . .

- **Integration with tactics for proving programs**
  Symbolic execution tactics for weakest preconditions
Scaling up

Iris Proof Mode scales and is used for any project involving Iris today because:

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

▶ **Support for advanced features of separation logic**
  Higher-order quantification, invariants, ghost state, later
  ⊿ modality, . . .

▶ **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
Implementation of separation logic tactics
How to embed a logic into a proof assistant?

<table>
<thead>
<tr>
<th>Deep embedding</th>
<th>Shallow embedding</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Inductive</strong> <code>form : Type :=</code></td>
<td><strong>Definition</strong> <code>iProp : Type :=</code></td>
</tr>
<tr>
<td></td>
<td>(* predicates over states *).</td>
</tr>
<tr>
<td>`</td>
<td>iAnd: form → form → form`</td>
</tr>
<tr>
<td>`</td>
<td>iForall: string → form → form → form`</td>
</tr>
<tr>
<td>`</td>
<td>iForall: ∀ A, (A → iProp) → iProp :=`</td>
</tr>
</tbody>
</table>
### How to embed a logic into a proof assistant?

<table>
<thead>
<tr>
<th>Deep embedding</th>
<th>Shallow embedding</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Inductive</strong> form : Type :=</td>
<td><strong>Definition</strong> iProp: Type :=</td>
</tr>
<tr>
<td></td>
<td>iAnd : form → form → form</td>
</tr>
<tr>
<td></td>
<td>iForall : string → form → form → form</td>
</tr>
<tr>
<td></td>
<td>(* semantic interpretation *).</td>
</tr>
<tr>
<td></td>
<td><strong>Definition</strong> iForall : ∀ A, (A → iProp) → iProp :=</td>
</tr>
<tr>
<td></td>
<td>(* semantic interpretation *).</td>
</tr>
</tbody>
</table>

- Traverse formulas using Coq functions (fast)
- Reflective tactics (fast)
- Traverse formulas on the meta level (slow)
- Tactics on the meta level (slow)
### How to embed a logic into a proof assistant?

<table>
<thead>
<tr>
<th>Deep embedding</th>
<th>Shallow embedding</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Inductive</strong> `form : Type :=**</td>
<td></td>
</tr>
<tr>
<td><em>iAnd : form → form → form</em></td>
<td></td>
</tr>
<tr>
<td><em>iForall : string → form → form → form</em></td>
<td><strong>Definition</strong> `iProp : Type :=**</td>
</tr>
<tr>
<td>(* predicates over states *)</td>
<td>(* semantic interpretation *)</td>
</tr>
<tr>
<td><strong>Definition</strong> <code>iAnd : iProp → iProp → iProp :=</code></td>
<td>(** semantic interpretation **)</td>
</tr>
<tr>
<td>(* semantic interpretation *)</td>
<td>(** semantic interpretation **)</td>
</tr>
<tr>
<td><strong>Definition</strong> <code>iForall : ∀ A, (A → iProp) → iProp :=</code></td>
<td></td>
</tr>
<tr>
<td>(* semantic interpretation *)</td>
<td></td>
</tr>
</tbody>
</table>

- Traverse formulas using Coq functions (fast)
- Reflective tactics (fast)
- Need to explicitly encode binders
- Need to embed features like lists
- 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?

<table>
<thead>
<tr>
<th>Deep embedding</th>
<th>Shallow embedding</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Inductive</strong> form : Type :=</td>
<td><strong>Definition</strong> iProp: Type :=</td>
</tr>
<tr>
<td></td>
<td>iAnd : form → form → form</td>
</tr>
<tr>
<td></td>
<td>iForall : string → form → form → form</td>
</tr>
<tr>
<td>(* semantic interpretation *)</td>
<td>(* semantic interpretation *).</td>
</tr>
<tr>
<td><strong>Definition</strong> iForall : ∀ A, (A → iProp) → iProp :=</td>
<td><strong>Definition</strong> iForall : ∀ A, (A → iProp) → iProp :=</td>
</tr>
<tr>
<td>(* semantic interpretation *).</td>
<td></td>
</tr>
</tbody>
</table>

- 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

- 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
How to embed a logic into a proof assistant?

<table>
<thead>
<tr>
<th>Deep embedding</th>
<th>Shallow embedding</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Inductive</strong> form : Type :=</td>
<td><strong>Definition</strong> iProp: Type :=</td>
</tr>
<tr>
<td></td>
<td>(* predicates over states *).</td>
</tr>
<tr>
<td></td>
<td>Definition iAnd: iProp → iProp → iProp :=</td>
</tr>
<tr>
<td></td>
<td>(* semantic interpretation *).</td>
</tr>
<tr>
<td></td>
<td>Definition iForall: ∀ A, (A → iProp) → iProp :=</td>
</tr>
<tr>
<td></td>
<td>(* semantic interpretation *).</td>
</tr>
</tbody>
</table>

- 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

| Traverse formulas on the meta level (slow) |
| Tactic 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 (1)

Lemma test \{A\} (P Q : iProp) (Ψ : A → iProp) :

\[ P \overset{\exists a, \Psi a}{} (\overset{\Psi a}{} Q) \overset{\exists a, P \overset{\Psi a}}{} . \]

Proof.

iIntros "[H1 [H2 H3]]".

iDestruct "H2" as (x) "H2".

iSplitL "H3".

- iAssumption.

- iExists x.

  iFrame.

Qed.
Deeply embedded contexts in IPM (1)

Lemma test \{A\} (P Q : iProp) (Ψ : A → iProp) :

\[
P \ast (\exists a, \Psi a) \ast Q \rightarrow Q \ast \exists a, P \ast \Psi a.
\]

Proof.
iIntros "[H1 [H2 H3]]".
iDestruct "H2" as (x) "H2".
iSplitL "H3".
- iAssumption.
- iExists x.
iFrame.
Qed.

1 subgoal
A : Type
P, Q : iProp
Ψ : A → iProp
x : A

\[
Q \ast (\exists a : A, P \ast \Psi a)
\]
Deeply embedded contexts in IPM (1)

Lemma test \{A\} (P Q : iProp) (Ψ : A → iProp) :
\[ P * (\exists a, \Psi a) * Q \rightarrow Q * \exists a, P * \Psi a. \]

Proof.
- iIntros "[H1 [H2 H3]]".
- iDestruct "H2" as (x) "H2".
- Unset Printing Notations.

1 subgoal
A : Type
P, Q : iProp
Ψ : A → iProp
x : A
"H1" : P
"H2" : Ψ x

Notation for deeply embedded context

Q * (\exists a : A, P * Ψ a)
Deeply embedded contexts in IPM (1)

Lemma test \{A\} (P Q : iProp) (Ψ : A → iProp) :
\[ P * (\exists a, Ψ a) * Q \quad Q * \exists a, P * Ψ a. \]

Proof.
\begin{itemize}
  \item iIntros "[H1 [H2 H3]]".
  \item iDestruct "H2" as (x) "H2".
  \item Unset Printing Notations.
\end{itemize}
Deeply embedded contexts in IPM (2)

Visible goal (with pretty printing):

\[ \overrightarrow{x} : \overrightarrow{\phi} \quad \text{Variables and pure Coq hypotheses} \]

\[ \Pi \quad \text{Spatial separation logic hypotheses} \]

\[ R \quad \text{Separation logic goal} \]
Deeply embedded contexts in IPM (2)

Visible goal (with pretty printing):
\[ \vec{x} : \vec{\phi} \quad \text{Variables and pure Coq hypotheses} \]
\[ \Pi \quad \text{Spatial separation logic hypotheses} \]
\[ R \quad \text{Separation logic goal} \]

Actual Coq goal (without pretty printing):
\[ \vec{x} : \vec{\phi} \]
\[ \Pi \vdash Q \]

Where:
\[ \Pi \vdash Q \triangleq \bigstar \Pi \vdash Q \]
Implementation of the iSplitL/iSplitR tactic

Tactics implemented by reflection as mere lemmas:

**Lemma** tac_sep_split \( \Pi_1 \Pi_2 \text{ lr js } Q_1 Q_2 : \)

\[
\text{envs_split lr js } \Pi = \text{Some } (\Pi_1,\Pi_2) \rightarrow \\
(\Pi_1 \vdash Q_1) \rightarrow (\Pi_2 \vdash Q_2) \rightarrow \Pi \vdash Q_1 \ast Q_2.
\]

\[\dfrac{\Pi_1 \vdash Q_1 \quad \Pi_2 \vdash Q_2}{\Pi_1,\Pi_2 \vdash Q_1 \ast Q_2}\]
Implementation of the iSplitL/iSplitR tactic

Tactics implemented by reflection as mere lemmas:

\[
\text{Lemma } \text{tac\_sep\_split } \Pi \Pi_1 \Pi_2 \text{ lr js } Q_1 Q_2 : \\
\Pi = \text{Some } (\Pi_1, \Pi_2) \rightarrow \\
(\Pi_1 \vdash Q_1) \rightarrow (\Pi_2 \vdash Q_2) \rightarrow \Pi \vdash Q_1 \ast Q_2.
\]

\[
\frac{\Pi_1 \vdash Q_1 \quad \Pi_2 \vdash Q_2}{\Pi_1, \Pi_2 \vdash Q_1 \ast Q_2}
\]

Context splitting implemented as a computable Coq function
Implementation of the iSplitL/iSplitR tactic

Tactics implemented by reflection as mere lemmas:

**Lemma** tac_sep_split \(\prod \prod_1 \prod_2 \text{lr \ js} \ Q_1 \ Q_2 :\)

\[
\text{envs\_split \ lr \ js \ \prod = \ Some \ (\prod_1,\prod_2) \rightarrow} \\
(\prod_1 \vdash Q_1) \rightarrow (\prod_2 \vdash Q_2) \rightarrow \prod \vdash Q_1 \ast Q_2.
\]

Context splitting implemented as a computable Coq function

Ltac wrappers around the reflective tactic:

**Tactic Notation** "iSplitL" constr(Hs) :=

let Hs := words Hs in
let Hs := eval vm_compute in (INamed \$> Hs) in
eapply tac_sep_split with _ _ Left Hs _ _;
[pm_reflexivity ||
  fail "iSplitL: hypotheses" Hs "not found"
| (* goal 1 *)
| (* goal 2 *)].
Implementation of the iSplitL/iSplitR tactic

Tactics implemented by reflection as mere lemmas:

**Lemma** tac_sep_split Π Π_1 Π_2 lr js Q_1 Q_2 :
envs_split lr js Π = Some (Π_1,Π_2) →
(Π_1 ⊩ Q_1) → (Π_2 ⊩ Q_2) → Π ⊩ Q_1 * Q_2.

\[
\begin{align*}
\Pi_1 & \vdash Q_1 & \Pi_2 & \vdash Q_2 \\
\Pi_1, \Pi_2 & \vdash Q_1 * Q_2
\end{align*}
\]

Context splitting implemented as a computable Coq function

Ltac wrappers around the reflective tactic:

**Tactic Notation** "iSplitL" constr(Hs) :=
let Hs := words Hs in
let Hs := eval vm_compute in (INamed <$> Hs) in
eapply tac_sep_split with _ _ Left Hs _ _;
[pm_reflexivity ||
fail "iSplitL: hypotheses" Hs "not found"
| (* goal 1 *)
| (* goal 2 *) ].

Report sensible error to the user
Proving program specifications
Proving program specifications

Consider:

\[ \{ x \mapsto v_1 \ast y \mapsto v_2 \} \text{swap}(x, y) \{ x \mapsto v_2 \ast y \mapsto v_1 \} \]

How to use IPM to manipulate the precondition?
Proving program specifications

Consider:

\[ \{ x \mapsto v_1 \ast y \mapsto v_2 \} \text{swap}(x, y) \{ x \mapsto v_2 \ast y \mapsto v_1 \} \]

How to use IPM to manipulate the precondition?

**Solution:** define Hoare triple in terms of weakest preconditions

We let:

\[ \{ P \} e \{ Q \} \triangleq \Box (P \rightarrow \text{wp } e \{ Q \}) \]

where \( \text{wp } e \{ Q \} \) gives the *weakest precondition* under which:

- all executions of \( e \) are safe
- the final state of \( e \) satisfies the postcondition \( Q \)
Proving swap using symbolic execution

Definition swap : val := \x : "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.
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 #l1
{{ _, l1 → v2 * l2 → v1 }}.

Proof.
iIntros "!# [Hl1 Hl2]".
do 2 wp_let.
wp_load; wp_let.
wp_load.
wp_store.
wp_store.
iFrame.
Qed.
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.

1 subgoal
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 := \( \lambda: \text{"x" "y"} \),

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.

1 subgoal
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 := \( \lambda: "x" "y" , \)

let: "tmp" := !"x" in

"x" ← !"y";;

"y" ← "tmp".

**Lemma** swap_spec l1 l2 v1 v2 :

\[
\{ \{ l1 ↦ v1 \land l2 ↦ v2 \} \} \text{ swap } \#l1 \#l2
\]

\[
\{ \_ , l1 ↦ v2 \land l2 ↦ v1 \} .
\]

**Proof.**

iIntros "!# [Hl1 Hl2]".

do 2 wp_let.

wp_load; wp_let.

wp_load.

wp_store.

wp_store.

iFrame.

Qed.

1 subgoal

l1, l2 : loc

v1, v2 : val

____________________________________(1/1)

"Hl1" : l1 ↦ v1

"Hl2" : l2 ↦ v2

___________________________*

WP

\#l1 ← ! \#l2 ;; \#l2 ← v1

\{ \_ , l1 ↦ v2 \land 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.
Qed.

1 subgoal
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.
   wp_store.
   iFrame.
Qed.

1 subgoal
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 := \( \lambda: "x" "y" \),
  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.

1 subgoal
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 := \( \lambda: "x" \ "y" \),

\begin{align*}
&\text{let: } "tmp" := !"x" \text{ in} \\
&"x" \leftarrow !"y" ; ; \\
&"y" \leftarrow "tmp".
\end{align*}

**Lemma** swap_spec l1 l2 v1 v2 :

\[
\begin{align*}
\{&\ l1 \mapsto v1 \ast l2 \mapsto v2 \} \quad \text{swap #l1 #l2} \\
\{&\ _, l1 \mapsto v2 \ast l2 \mapsto v1 \}\}.
\end{align*}
\]

**Proof**.

\begin{align*}
i\text{Intros } "!# \ [Hl1 Hl2]". \\
do 2 \ wp_{let}. \\
wp\_load; \ wp_{let}. \\
wp\_load. \\
wp\_store. \\
wp\_store. \\
iFrame. \\
\text{Qed.}
\end{align*}

No more subgoals.
Projects that use IPM

The approach in Iris Proof Mode actually scales beyond swap

- Non-determinism in C [Frumin et al.; ESOP'19]
- Time complexity [Mével et al.; ESOP’19]
- Resource obligations [Bizjak et al.; POPL’19]
- The Rust type system [Jung et al.; POPL’18]
- Haskell’s ST monad [Timany et al.; POPL’18]
- Program refinements [Tassarotti et al.; ESOP’17 & Frumin et al.; LICS’18]
- Iris’s meta theory [Krebbers et al.; ESOP’17]
- Weak memory [Kaiser et al.; ECOOP’17]
- Object capabilities [Swasey et al.; OOPSLA’17]
Next step: Going beyond Iris
Making Iris Proof Mode independent of Iris

It sounds easy [Krebbers et al.; POPL'17]:

[...] we believe that our proof mode is very generic, and can be applied to a variety of different embedded logics [...]
Making Iris Proof Mode independent of Iris

It sounds easy [Krebbers et al.; POPL’17]:

[...] we believe that our proof mode is very generic, and can be applied to a variety of different embedded logics [...]
Problem #1: Iris propositions are affine

In Iris you may “forget” about resources:

\[ \{ x \mapsto v_1 \ast y \mapsto v_2 \} \ x, y \mapsto v_1 \]
Problem #1: Iris propositions are affine

In Iris you may “forget” about resources:

\[ \{ x \mapsto v_1 \ast y \mapsto v_2 \} \text{swap}(x, y) \{ y \mapsto v_1 \} \]

Due to the affinity axiom \( P \ast Q \vdash Q \), which is hard-wired into many tactics:

\[\begin{align*}
\text{iClear} & \\
\text{iAssumption} & \\
\Pi, P & \vdash P
\end{align*}\]
Problem #1: Iris propositions are affine

In Iris you may “forget” about resources:

\[
\{ x \mapsto v_1 \times y \mapsto v_2 \} \; swap(x, y) \{ y \mapsto v_1 \}
\]

Due to the **affinity axiom** \( P \times Q \vdash Q \), which is hard-wired into many tactics:

- iClear
  \[ \Pi \vdash Q \]
  \[ \Pi, P \vdash Q \]
- iAssumption
  \[ \Pi, P \vdash P \]

**Not having the affinity axiom is useful:** precise accounting of resources

**Challenge:** How to disentangle the affinity axiom from the Iris tactics?
Problem #2: No tactical support for derived logics

Coq (Prop)

propositions defined in terms of

Iris (iProp)

Proof using standard Coq tactics

Proof using Iris tactics
Problem #2: No tactical support for derived logics

Coq (Prop) → Iris (iProp) → Derived logic (e.g. iGpsProp)

- Proof using standard Coq tactics
- Proof using Iris tactics

propositions defined in terms of, iGpsProp $\triangleq$ View $\xrightarrow{\text{mon}}$ iProp
Problem #2: No tactical support for derived logics

- Coq (Prop)
  - Proof using standard Coq tactics

- Iris (iProp)
  - Proof using Iris tactics
  - Propositions defined in terms of $\text{iGpsProp} \triangleq \text{View} \xrightarrow{\text{mon}} \text{iProp}$

- Derived logic (e.g. iGpsProp)
  - Proof using ???
Problem #2: No tactical support for derived logics

Derived logic (e.g. iGpsProp)  
\[ \text{propositions defined in terms of, } iGpsProp \triangleq \text{View} \xrightarrow{\text{mon}} \text{iProp} \]

Coq (Prop)  
Proof using standard Coq tactics

Iris (iProp)  
Proof using Iris tactics

Challenge: How to reason in logics defined in terms of another

New features w.r.t. Iris Proof Mode:
- MoSeL is parameterized by a general abstraction of separation logic
- MoSeL supports general and affine separation logics, and combinations thereof
- MoSeL supports reasoning in derived separation logics
- MoSeL can be fine-tuned for each logic using type classes

New features w.r.t. Iris Proof Mode:

▶ MoSeL is parameterized by a general abstraction of separation logic
▶ MoSeL supports general and affine separation logics, and combinations thereof
▶ MoSeL supports reasoning in derived separation logics
▶ MoSeL can be fine-tuned for each logic using type classes

MoSeL is usable in practice: we used it on very different existing separation logics

CFML CHL Fairis iGPS Iris Iron
Setup of MoSeL

Instance of a separation logic abstraction

Type class instances to fine tune tactics

MoSeL’s tactics
Instance of a separation logic abstraction

Type class instances to fine tune tactics

MoSeL’s tactics
How to abstract over separation logics?
BI logics: An abstract interface for separation logics

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

- Operations True, False, \(\wedge, \lor, \Rightarrow, \forall, \exists\) satisfying the axioms of intuitionistic logic
- Operations \(\mathsf{emp}, *, \neg\ast\) satisfying:

\[
\begin{align*}
\mathsf{emp} * P & \not\vdash P \\
P * Q & \vdash Q * P \\
(P * Q) * R & \vdash P * (Q * R)
\end{align*}
\]

\[
\begin{align*}
P_1 \vdash Q_1 & \quad P_2 \vdash Q_2 \\
& \quad (P_1 * P_2) \vdash Q_1 * Q_2 \\
P \vdash Q & \not\vdash R \\
& \quad P * Q \vdash R \\
& \quad P \vdash Q \not\vdash R
\end{align*}
\]
BI logics: An abstract interface for separation logics

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

- Operations \(\text{True}, \text{False}, \land, \lor, \Rightarrow, \forall, \exists\) satisfying the axioms of intuitionistic logic
- Operations \(\text{emp}, \ast, \ast\) satisfying:

\[
\begin{align*}
\text{emp} \ast P & \vdash P \\
P \ast Q & \vdash Q \ast P \\
(P \ast Q) \ast R & \vdash P \ast (Q \ast R)
\end{align*}
\]

\[
\begin{align*}
P_1 \vdash Q_1 & \quad P_2 \vdash Q_2 \\
\frac{P_1 \ast P_2 \vdash Q_1 \ast Q_2}{P \vdash Q \ast R}
\end{align*}
\]

Structure \(\text{bi} := \text{Bi}\{\)

- \(\text{bi} \_\text{car} \Rightarrow \text{Type}\)
- \(\text{bi} \_\text{entails} : \text{bi} \_\text{car} \rightarrow \text{bi} \_\text{car} \rightarrow \text{Prop}\)
- \(\text{bi} \_\text{forall} : \forall A, (A \rightarrow \text{bi} \_\text{car}) \rightarrow \text{bi} \_\text{car}\)
- \(\text{bi} \_\text{sep} : \text{bi} \_\text{car} \rightarrow \text{bi} \_\text{car} \rightarrow \text{bi} \_\text{car}\)

(* other separation logic operators and axioms *)

\}.
Proofs in MoSeL

Proofs in a specific logic:

Lemma test\{A\}(PQ:iGpsProp)(Ψ:A→iGpsProp):
P∗(∃a,Ψa)∗Q→∗Q∗∃a,P∗Ψa.
Proof.
iIntros "[H1 [H2 H3]]".
iDestruct "H2" as (x) "H2".
iSplitL "H3".
− iAssumption.
− iExists x.
iFrame.
Qed.

Proofs for all logics:

P∗(∃a,Ψa)∗Q→∗Q∗∃a,P∗Ψa.
Proof.
iIntros "[H1 [H2 H3]]".
iDestruct "H2" as (x) "H2".
iSplitL "H3".
− iAssumption.
− iExists x.
iFrame.
Qed.

Lemma for another logic than Iris

Lemma universally quantified in the BI logic
Proofs in MoSeL

Proofs in a specific logic:

Lemma test\{A\}(PQ : iGpsProp)(Ψ : A → iGpsProp) :
P * (∃ a, Ψ a) * Q → Q * ∃ a, P * Ψ a.

Proof.
iIntros "[H1 [H2 H3]]".
iDestruct "H2" as (x) "H2".
iSplitL "H3".
iAssumption.
iFrame.
Qed.

Proofs for all logics:

Lemma test\{PROP : bi\}{A}(PQ : PROP)(Ψ : A → PROP) :
P * (∃ a, Ψ a) * Q → Q * ∃ a, P * Ψ a.

Proof.
iIntros "[H1 [H2 H3]]".
iDestruct "H2" as (x) "H2".
iSplitL "H3".
iAssumption.
iFrame.
Qed.

Lemma for another logic than Iris:
Proofs in MoSeL

Proofs in a specific logic:

Lemma test\{A\}(P \land Q : \Pi \rightarrow \iGpProp)(\Psi : A \rightarrow \iGpProp) :

\[
P \land (\exists \ a, \Psi \ a) \land Q \land \exists \ a, P \land \Psi \ a.
\]

Proof.

iIntros "[H1 [H2 H3]]".

iAssumption.

− iExists x.

iFrame.

Qed.

Proofs for all logics:

Lemma test\{PROP : bi\}{A\}(P \land Q : PROP)(\Psi : A \rightarrow PROP) :

\[
P \land (\exists \ a, \Psi \ a) \land Q \land \exists \ a, P \land \Psi \ a.
\]

Proof.

iIntros "[H1 [H2 H3]]".

Destruct "H2" as (x) "H2".

SplitL "H3".

iAssumption.

− iExists x.

iFrame.

Qed.

Lemma for another logic than Iris

Lemma universally quantified in the BI logic
Addressing challenge #1:
Disentangling the affinity axiom
A poor man’s solution

Make two versions of the tactics

1. For affine logics (like Iris and iGPS)
2. For non-affine logics (like CFML and CHL)

Problems:
- Duplicate work/maintenance
- Some logics mix affine and non-affine propositions, for example:
  - GC locations
  - Non-GC locations
  - \( \ell \mapsto \text{gc} \)
  - \( \ell \mapsto \text{v} \)

(Another example in [Tassarotti et al.; ESOP’17])
A poor man’s solution

**Make two versions of the tactics**

1. For **affine logics** (like Iris and iGPS)
2. For **non-affine logics** (like CFML and CHL)

**Problems:**

- Duplicate work/maintenance
- Some logics mix affine and non-affine propositions, for example:

  \[
  \begin{align*}
  \textbf{GC locations} & \quad (\text{affine}) \\
  \ell & \mapsto \text{gc } v
  \end{align*}
  \]
  \[
  \begin{align*}
  \textbf{Non-GC locations} & \quad (\text{not affine}) \\
  \ell & \mapsto v
  \end{align*}
  \]

  (Another example in [Tassarotti et al.; ESOP’17])
Key idea

▷ **Don’t**: classify whether the whole logic is affine

▷ **Do**: classify whether individual propositions are affine
Classifying whether propositions are affine

Affine propositions:

\[
\text{affine}(P) \triangleq P \vdash \text{emp} \quad \text{(propositions that can be “thrown away”)}
\]

The new tactics:

\[
\text{iClear} \\
\Pi \vdash Q \quad \text{affine}(P) \\
\Pi, P \vdash Q
\]

\[
\text{iAssumption} \\
\text{affine}(\Pi) \\
\Pi, Q \vdash Q
\]
Classifying whether propositions are affine in Coq

A new type class:

Class Affine {PROP : bi} (Q : PROP) := affine : Q ⊢ emp.

Instances:

- Tell MoSeL that specific connectives are affine:

  Instance mapsto_gc_affine l v : Affine (l ↦ gc v).

- Capture that affine propositions are closed under most connectives:

  Instance sep_affine {PROP : bi} (P Q : bi) :
  Affine P → Affine Q → Affine (P * Q).
MoSeL: A General, Extensible Modal Framework for Interactive Proofs in Separation Logic in Coq

What about modalities?
The affine modality

The **affine modality**: 

\[
\langle \text{affine} \rangle P \triangleq P \land \text{emp}
\]

\[
\cong "P \text{ holds using just affine resources}"
\]

This modality is useful for reasoning about affinity
The affine modality

The **affine modality**:

\[
\langle \text{affine} \rangle P \triangleq P \land \text{emp} \\
\approx \text{“}P\text{ holds using just affine resources”}
\]

This modality is useful for reasoning about affinity

- Can be used to turn any proposition into an affine version, *e.g.*
  - A wand that can be dropped \( \langle \text{affine} \rangle (P \rightarrow Q) \)
The affine modality

The **affine modality**:

\[ \langle \text{affine} \rangle P \triangleq P \land \text{emp} \]
\[ \approx \text{"} P \text{ holds using just affine resources"} \]

This modality is useful for reasoning about affinity

- Can be used to turn any proposition into an affine version, *e.g.*
  A wand that can be dropped \( \langle \text{affine} \rangle (P \rightarrow Q) \)

- Commutes with most operators, *e.g.*
  \( \langle \text{affine} \rangle (P \lor Q) \vdash \langle \text{affine} \rangle P \lor \langle \text{affine} \rangle Q \)
The affine modality

The **affine modality**:

\[
\langle \text{affine} \rangle P \triangleq P \land \text{emp}
\]

\[
\approx \quad \text{“} P \text{ holds using just affine resources”}
\]

This modality is useful for reasoning about affinity

- Can be used to turn any proposition into an affine version, e.g.
  A wand that can be dropped \( \langle \text{affine} \rangle (P \rightarrow Q) \)

- Commutes with most operators, e.g.
  \( \langle \text{affine} \rangle (P \lor Q) \vdash \Dash \langle \text{affine} \rangle P \lor \langle \text{affine} \rangle Q \)

- Gives rise to an alternative classification of affine propositions
  \[\text{affine}(P) \quad \text{iff} \quad P \vdash \langle \text{affine} \rangle P\]
The idea of carving out classes of propositions and defining their corresponding modalities is widely applicable:

- Persistent propositions
- Intuitionistic propositions
- Absorbing propositions
- Timeless propositions (in step-indexed logics)
- Objective propositions (in iGPS)
- Normal propositions (in CFML)
- ... 

The ICFP'18 paper shows how to modularly deal with such classes and use them in general tactics.
Thanks to

My coauthors on the IPM/MoSeL papers:

Ralf Jung  Jacques-Henri Jourdan  Amin Timany  Joseph Tassarotti
Jan-Oliver Kaiser  Arthur Charguéraud  Derek Dreyer  Lars Birkedal

And all that have contributed by using IPM/MoSeL in one way or another
Thank you!

Download MoSeL/Iris at http://iris-project.org/

**Advertisement.** I am currently looking for a PhD student (4 years)

**Topics:** Separation logic for multilingual programs, asynchronous I/O, non-functional properties, verified compilation, ...  

Interested/Know someone? Get in touch!