Extending the Iris Proof Mode with Inductive Predicates using Elpi

Luko van der Maas

Computing Science
Radboud University

Program verification

- Verify programs by specifying pre and post conditions

Program verification

- Verify programs by specifying pre and post conditions
- Specification happens in separation logic

Program verification

- Verify programs by specifying pre and post conditions
- Specification happens in separation logic
- We make use of embeddings of separation logic in Coq

Program verification

- Verify programs by specifying pre and post conditions
- Specification happens in separation logic
- We make use of embeddings of separation logic in Coq
- Iris 2015 (Jung, Krebbers, et. al.)

Separation logic with Hoare triples

$$[\mathrm{isD}\ d\ y]\ \mathrm{op}\ d\ x\ [\mathrm{isD}\ d\ (\mathrm{f}\ x\ y)]$$

Separation logic with Hoare triples

$$[\mathrm{isD}\ d\ y]\ \mathrm{op}\ d\ x\ [\mathrm{isD}\ d\ (\mathrm{f}\ x\ y)]$$

Imperative program

Separation logic with Hoare triples

$$[\mathrm{isD}\ d\ y]\ \mathrm{op}\ d\ x\ [\mathrm{isD}\ d\ (\mathrm{f}\ x\ y)]$$

Imperative program

Functional program

Separation logic with Hoare triples

Representation predicate

$[\mathrm{isD}\ d\ y]\ \mathrm{op}\ d\ x\ [\mathrm{isD}\ d\ (\mathrm{f}\ x\ y)]$

Imperative program

Functional program

Separation logic with Hoare triples

Representation predicate

[isD d y] op d x [isD d (f x y)]

Imperative program

Functional program

Separation logic with Hoare triples

Representation predicate

$$[\mathrm{isD}\ d\ y]\ \mathrm{op}\ d\ x\ [\mathrm{isD}\ d\ (\mathrm{f}\ x\ y)]$$

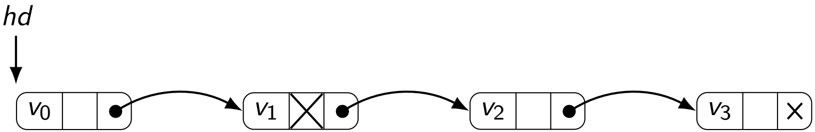
Imperative program

Functional program

$$[\mathrm{isList}\ hd\ \vec{v}]\ \ \mathrm{delete}\ \ hd\ i\ [\mathrm{isList}\ hd\ (\mathrm{remove}\ i\ \vec{v})]$$

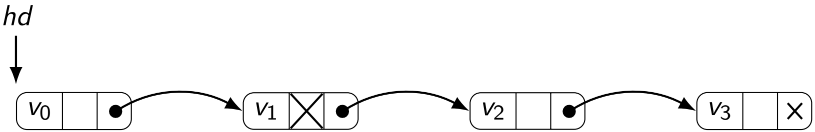
Representation predicates

Representation predicates



Harris (2001)

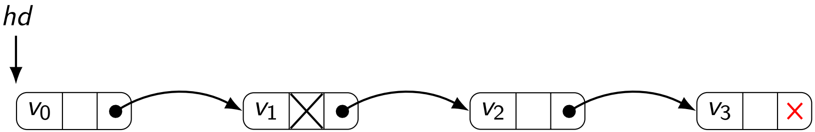
Representation predicates



Harris (2001)

$$\begin{aligned}\mathrm{isMLL}\ hd\ \vec{v} = \ & (hd = \mathbf{none} * \vec{v} = []) \lor \\ & (\exists \ell, v', tl.\ hd = \mathbf{some}\ \ell * \ell \mapsto (v', \mathbf{true}, tl) * \mathrm{isMLL}\ tl\ \vec{v}) \lor \\ & \left(\begin{array}{c} \exists \ell, v', \vec{v}'', tl.\ hd = \mathbf{some}\ \ell * \ell \mapsto (v', \mathbf{false}, tl) * \\ \vec{v} = v' :: \vec{v}'' * \mathrm{isMLL}\ tl\ \vec{v}'' \end{array} \right)\end{aligned}$$

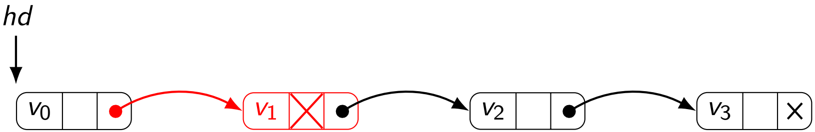
Representation predicates



Harris (2001)

$$
\begin{aligned}
\mathsf{isMLL}\; hd\; \vec{v} = \;& (hd = \mathbf{none} * \vec{v} = []) \lor \\
& (\exists \ell, v', tl.\; hd = \mathtt{some}\, \ell * \ell \mapsto (v', \mathtt{true}, tl) * \mathsf{isMLL}\; tl\; \vec{v}) \lor \\
& \left(\begin{array}{c} \exists \ell, v', \vec{v}'', tl.\; hd = \mathtt{some}\, \ell * \ell \mapsto (v', \mathtt{false}, tl) * \\ \vec{v} = v' :: \vec{v}'' * \mathsf{isMLL}\; tl\; \vec{v}'' \end{array} \right)
\end{aligned}
$$

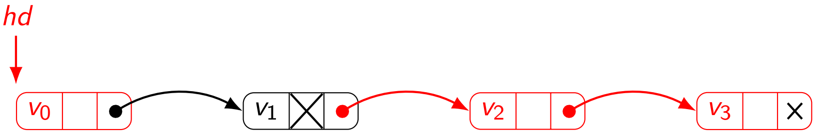
Representation predicates



Harris (2001)

$$
\begin{aligned}
\mathrm{isMLL}\ hd\ \vec{v} = \ & (hd = \mathbf{none} * \vec{v} = []) \lor \\
& (\exists \ell, v', tl.\ hd = \mathtt{some}\ \ell * \ell \mapsto (v', \mathtt{true}, tl) * \mathrm{isMLL}\ tl\ \vec{v}) \lor \\
& \begin{pmatrix} \exists \ell, v', \vec{v}'', tl.\ hd = \mathtt{some}\ \ell * \ell \mapsto (v', \mathtt{false}, tl) * \\ \vec{v} = v' :: \vec{v}'' * \mathrm{isMLL}\ tl\ \vec{v}'' \end{pmatrix}
\end{aligned}
$$

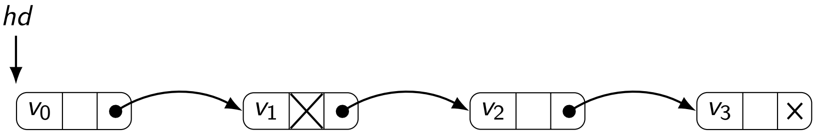
Representation predicates



Harris (2001)

$$
\begin{aligned}
\text{isMLL}\ hd\ \vec{v} = \ & (hd = \textbf{none} * \vec{v} = []) \vee \\
& (\exists \ell, v', tl.\ hd = \textbf{some}\ \ell * \ell \mapsto (v', \texttt{true}, tl) * \text{isMLL}\ tl\ \vec{v}) \vee \\
& \left(\begin{array}{c} \exists \ell, v', \vec{v}'', tl.\ hd = \textbf{some}\ \ell * \ell \mapsto (v', \texttt{false}, tl) * \\ \vec{v} = v' :: \vec{v}'' * \text{isMLL}\ tl\ \vec{v}'' \end{array} \right)
\end{aligned}
$$

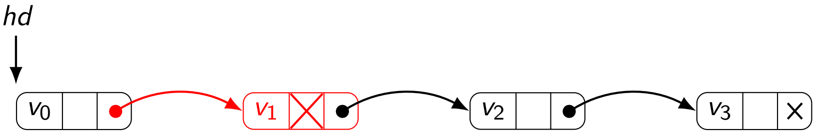
Representation predicates



Harris (2001)

$$
\begin{aligned}
\text{isMLL}\ hd\ \vec{v} = \ & (hd = \textbf{none} * \vec{v} = []) \lor \\
& (\exists \ell, v', tl.\ hd = \textbf{some}\ \ell * \ell \mapsto (v', \texttt{true}, tl) * \text{isMLL}\ tl\ \vec{v}) \lor \\
& \left(\begin{array}{c} \exists \ell, v', \vec{v}'', tl.\ hd = \textbf{some}\ \ell * \ell \mapsto (v', \texttt{false}, tl) * \\ \vec{v} = v' :: \vec{v}'' * \text{isMLL}\ tl\ \vec{v}'' \end{array} \right)
\end{aligned}
$$

Representation predicates



Harris (2001)

$$
\begin{aligned}
\text{isMLL}\ hd\ \vec{v} = \ & (hd = \textbf{none} * \vec{v} = []) \lor \\
& (\exists \ell, v', tl.\ hd = \textbf{some}\ \ell * \ell \mapsto (v', \textbf{true}, tl) * \text{isMLL}\ tl\ \vec{v}) \lor \\
& \left(\begin{array}{c} \exists \ell, v', \vec{v}'', tl.\ hd = \textbf{some}\ \ell * \ell \mapsto (v', \textbf{false}, tl) * \\ \vec{v} = v' :: \vec{v}'' * \text{isMLL}\ tl\ \vec{v}'' \end{array} \right)
\end{aligned}
$$

Outline of our solution

Coq

```
eiInd
Inductive is_MLL : val → list val → iProp :=
    | empty_is_MLL : is_MLL NONEV []
    | mark_is_MLL v vs l tl :
      l ↦ (v, #true, tl) -∗ is_MLL tl vs -∗
      is_MLL (SOMEV #l) vs
    | cons_is_MLL v vs tl l :
      l ↦ (v, #false, tl) -∗ is_MLL tl vs -∗
      is_MLL (SOMEV #l) (v :: vs).
```

Outline of our solution

```
eiInd
Inductive is_MLL : val → list val → iProp :=
    | empty_is_MLL : is_MLL NONEV []
    | mark_is_MLL v vs l tl :
      l ↦ (v, #true, tl) -∗ is_MLL tl vs -∗
      is_MLL (SOMEV #l) vs
    | cons_is_MLL v vs tl l :
      l ↦ (v, #false, tl) -∗ is_MLL tl vs -∗
      is_MLL (SOMEV #l) (v :: vs).
```

Coq

- Definition of `is_MLL`

Outline of our solution

```
eiInd
Inductive is_MLL : val → list val → iProp :=
    | empty_is_MLL : is_MLL NONEV []
    | mark_is_MLL v vs l tl :
      l ↦ (v, #true, tl) -∗ is_MLL tl vs -∗
      is_MLL (SOMEV #l) vs
    | cons_is_MLL v vs tl l :
      l ↦ (v, #false, tl) -∗ is_MLL tl vs -∗
      is_MLL (SOMEV #l) (v :: vs).
```

- Definition of `is_MLL`
- Proof of constructors, `empty_is_MLL`, `mark_is_MLL`, `cons_is_MLL`

Outline of our solution

```
eiInd
Inductive is_MLL : val → list val → iProp :=
    | empty_is_MLL : is_MLL NONEV []
    | mark_is_MLL v vs l tl :
        l ↦ (v, #true, tl) -∗ is_MLL tl vs -∗
        is_MLL (SOMEV #l) vs
    | cons_is_MLL v vs tl l :
        l ↦ (v, #false, tl) -∗ is_MLL tl vs -∗
        is_MLL (SOMEV #l) (v :: vs).
```

Coq

- Definition of `is_MLL`
- Proof of constructors, `empty_is_MLL`, `mark_is_MLL`, `cons_is_MLL`
- Proof of induction principle

Outline of our solution

```
eiInd                                                    Coq
Inductive is_MLL : val → list val → iProp :=
     | empty_is_MLL : is_MLL NONEV []
     | mark_is_MLL v vs l tl :
       l ↦ (v, #true, tl) -∗ is_MLL tl vs -∗
       is_MLL (SOMEV #l) vs
     | cons_is_MLL v vs tl l :
       l ↦ (v, #false, tl) -∗ is_MLL tl vs -∗
       is_MLL (SOMEV #l) (v :: vs).
```

- Definition of `is_MLL`
- Proof of constructors, `empty_is_MLL`, `mark_is_MLL`, `cons_is_MLL`
- Proof of induction principle
- Integration with IPM tactics

Approach

Theory

Challenges in practice

Approach

Theory

- Define the pre fixpoint function

Challenges in practice

Approach

Theory

- Define the pre fixpoint function
- Prove monotonicity

Challenges in practice

Approach

Theory

- Define the pre fixpoint function
- Prove monotonicity
- Apply least fixpoint theorem

Challenges in practice

Approach

Theory

- Define the pre fixpoint function
- Prove monotonicity
- Apply least fixpoint theorem

Challenges in practice

- Deal with n-ary predicates

Approach

Theory

- Define the pre fixpoint function
- Prove monotonicity
- Apply least fixpoint theorem

Challenges in practice

- Deal with n-ary predicates
- Proof search for monotonicity

Approach

Theory

- Define the pre fixpoint function
- Prove monotonicity
- Apply least fixpoint theorem

Challenges in practice

- Deal with *n*-ary predicates
- Proof search for monotonicity
- Integrating resulting definitions and lemmas into the Iris tactics language

Monotone pre fixpoint function

$$
\begin{aligned}
\text{isMLL}\ hd\ \vec{v} = \ & (hd = \mathbf{none} * \vec{v} = []) \lor \\
& (\exists \ell, v', tl.\ hd = \mathbf{some}\ l * l \mapsto (v', \mathbf{true}, tl) * \text{isMLL}\ tl\ \vec{v}) \lor \\
& \begin{pmatrix} \exists \ell, v', \vec{v}'', tl.\ hd = \mathbf{some}\ l * l \mapsto (v', \mathbf{false}, tl) * \\ \vec{v} = v' :: \vec{v}'' * \text{isMLL}\ tl\ \vec{v}'' \end{pmatrix}
\end{aligned}
$$

Monotone pre fixpoint function

$$
\begin{aligned}
\mathsf{isMLL}_{\mathsf{F}}\,\Phi\,hd\,\vec{v} = \; & (hd = \mathbf{none} * \vec{v} = []) \lor \\
& (\exists \ell, v, tl.\; hd = \mathbf{some}\, l * l \mapsto (v, \mathbf{true}, tl) * \Phi\, tl\, \vec{v}) \lor \\
& \left(\begin{array}{c} \exists \ell, v, \vec{v}'', tl.\; hd = \mathbf{some}\, l * l \mapsto (v, \mathbf{false}, tl) * \\ \vec{v} = v :: \vec{v}'' * \Phi\, tl\, \vec{v}'' \end{array} \right)
\end{aligned}
$$

Monotone pre fixpoint function

$$\begin{aligned}\mathrm{isMLL}_{\mathsf{F}}\,\Phi\,hd\,\vec{v} = \;& (hd = \mathbf{none} * \vec{v} = []) \lor \\ & (\exists \ell, v\prime, tl.\ hd = \mathbf{some}\,l * l \mapsto (v\prime, \mathtt{true}, tl) * \Phi\,tl\,\vec{v}) \lor \\ & \left(\begin{array}{c}\exists \ell, v\prime, \vec{v}'', tl.\ hd = \mathbf{some}\,l * l \mapsto (v\prime, \mathtt{false}, tl) * \\ \vec{v} = v\prime :: \vec{v}'' * \Phi\,tl\,\vec{v}''\end{array}\right)\end{aligned}$$

Definition (Monotonicity)

Function $\mathsf{F} \colon (A \to B \to iProp) \to A \to B \to iProp$ is *monotone* when, for any $\Phi, \Psi \colon A \to B \to iProp$, it holds that

$$\Box(\forall x, y.\, \Phi\,x\,y \twoheadrightarrow \Psi\,x\,y) \vdash \forall x, y.\, \mathsf{F}\,\Phi\,x\,y \twoheadrightarrow \mathsf{F}\,\Psi\,x\,y$$

Monotone signatures

Definition (Respectful relation)

$$R \Longrightarrow R' \triangleq \lambda f, g.\ \forall x, y.\ R\,x\,y \twoheadrightarrow\!\!* \ R'\,(f\,x)\,(g\,y)$$

Definition (Pointwise relation)

$$> R \triangleq \lambda f, g.\ \forall x.\ R\,(f\,x)\,(g\,x)$$

(Sozeau 2009)

Connective	Type	Signature
$*$	$iProp \to iProp \to iProp$	$(\twoheadrightarrow\!\!*) \Longrightarrow (\twoheadrightarrow\!\!*) \Longrightarrow (\twoheadrightarrow\!\!*)$

Monotone signatures

Definition (Respectful relation)

$$R \Longrightarrow R' \triangleq \lambda f, g.\ \forall x, y.\ R\,x\,y \twoheadrightarrow R'\,(f\,x)\,(g\,y)$$

Definition (Pointwise relation)

$$\gtrdot R \triangleq \lambda f, g.\ \forall x.\ R\,(f\,x)\,(g\,x)$$

(Sozeau 2009)

Connective	Type	Signature
$*$	$iProp \to iProp \to iProp$	$(\twoheadrightarrow) \Longrightarrow (\twoheadrightarrow) \Longrightarrow (\twoheadrightarrow)$
$\vee$	$iProp \to iProp \to iProp$	$(\twoheadrightarrow) \Longrightarrow (\twoheadrightarrow) \Longrightarrow (\twoheadrightarrow)$

Monotone signatures

Definition (Respectful relation)

$$R \Longrightarrow R' \triangleq \lambda f, g.\, \forall x, y.\, R\,x\,y \mathrel{-\!\!*} R'\,(f\,x)\,(g\,y)$$

Definition (Pointwise relation)

$$\geqslant\!\! R \triangleq \lambda f, g.\, \forall x.\, R\,(f\,x)\,(g\,x)$$

(Sozeau 2009)

Connective	Type	Signature
$*$	$iProp \to iProp \to iProp$	$(\mathrel{-\!\!*}) \Longrightarrow (\mathrel{-\!\!*}) \Longrightarrow (\mathrel{-\!\!*})$
$\vee$	$iProp \to iProp \to iProp$	$(\mathrel{-\!\!*}) \Longrightarrow (\mathrel{-\!\!*}) \Longrightarrow (\mathrel{-\!\!*})$
$\exists$	$(A \to iProp) \to iProp$	$\geqslant\!(\mathrel{-\!\!*}) \Longrightarrow (\mathrel{-\!\!*})$

Monotone signatures

Definition (Respectful relation)

$$R \Longrightarrow R' \triangleq \lambda f, g.\, \forall x, y.\, R\,x\,y \twoheadrightarrow R'\,(f\,x)\,(g\,y)$$

Definition (Pointwise relation)

$$\gtrdot R \triangleq \lambda f, g.\, \forall x.\, R\,(f\,x)\,(g\,x)$$

(Sozeau 2009)

Connective	Type	Signature
$*$	$iProp \to iProp \to iProp$	$(\twoheadrightarrow) \Longrightarrow (\twoheadrightarrow) \Longrightarrow (\twoheadrightarrow)$
$\vee$	$iProp \to iProp \to iProp$	$(\twoheadrightarrow) \Longrightarrow (\twoheadrightarrow) \Longrightarrow (\twoheadrightarrow)$
$\exists$	$(A \to iProp) \to iProp$	$\gtrdot(\twoheadrightarrow) \Longrightarrow (\twoheadrightarrow)$

Definition (Proper element of a relation)

Given a relation R: $iRel\ A$ and an element $x \in A$, x is a proper element of R if $R\,x\,x$.

$$((\twoheadrightarrow) \Longrightarrow (\twoheadrightarrow) \Longrightarrow (\twoheadrightarrow))(*)(*) =$$
$$\forall P, P'.\, (P \twoheadrightarrow P') \twoheadrightarrow \forall Q, Q'.\, (Q \twoheadrightarrow Q') \twoheadrightarrow (P * Q) \twoheadrightarrow (P' * Q')$$

Proof search

$$\square\,(\forall hd\,\vec{v}.\,\Phi\;hd\;\vec{v} \mathrel{-\!\!*} \Psi\;hd\;\vec{v}) \mathrel{-\!\!*}$$
$$\left(\begin{array}{ll}\forall hd\,\vec{v}. & \mathsf{isMLL_F}\;\Phi\;hd\;\vec{v} \mathrel{-\!\!*} \\ & \mathsf{isMLL_F}\;\Psi\;hd\;\vec{v}\end{array}\right)$$

Normalization

- Introduce quantifiers and modalities
  $\rightarrow$ Application step

Application

- Apply reflexivity
- Apply assumption
- Apply signature $\rightarrow$ Normalization step

Proof search

$$
\begin{array}{l}
(hd = \textbf{none} * \vec{v} = []) \lor \\
\left(\begin{array}{l} \exists \ell, v', tl. hd = \textbf{some}\ l\, * \\ \qquad l \mapsto (v', \texttt{true}, tl) * \\ \qquad \Phi\ tl\ \vec{v} \end{array} \right) \lor \\
\left(\begin{array}{l} \exists \ell, v', \vec{v}'', tl.\ hd = \textbf{some}\ l\, * \\ \qquad l \mapsto (v', \texttt{false}, tl) * \\ \qquad \vec{v} = v' :: \vec{v}'' * \Phi\ tl\ \vec{v}'' \end{array} \right)
\end{array}
$$

$$-\!\!*$$

$$
\begin{array}{l}
(hd = \textbf{none} * \vec{v} = []) \lor \\
\left(\begin{array}{l} \exists \ell, v', tl. hd = \textbf{some}\ l\, * \\ \qquad l \mapsto (v', \texttt{true}, tl) * \\ \qquad \Psi\ tl\ \vec{v} \end{array} \right) \lor \\
\left(\begin{array}{l} \exists \ell, v', \vec{v}'', tl.\ hd = \textbf{some}\ l\, * \\ \qquad l \mapsto (v', \texttt{false}, tl) * \\ \qquad \vec{v} = v' :: \vec{v}'' * \Psi\ tl\ \vec{v}'' \end{array} \right)
\end{array}
$$

Normalization

- Introduce quantifiers and modalities
  $\rightarrow$ Application step

Application

- Apply reflexivity
- Apply assumption
- Apply signature $\rightarrow$ Normalization step

Proof search

$$(hd = \textbf{none} * \vec{v} = []) \lor \left(\begin{array}{l} \exists \ell, v', tl. hd = \textbf{some}\ l * \\ \qquad l \mapsto (v', \texttt{true}, tl) * \\ \qquad \Phi\ tl\ \vec{v} \end{array} \right) \lor \left(\begin{array}{l} \exists \ell, v', \vec{v}'', tl.\ hd = \textbf{some}\ l * \\ \qquad l \mapsto (v', \texttt{false}, tl) * \\ \qquad \vec{v} = v' :: \vec{v}'' * \Phi\ tl\ \vec{v}'' \end{array} \right)$$

$$-\!\!*$$

$$(hd = \textbf{none} * \vec{v} = []) \lor \left(\begin{array}{l} \exists \ell, v', tl. hd = \textbf{some}\ l * \\ \qquad l \mapsto (v', \texttt{true}, tl) * \\ \qquad \Psi\ tl\ \vec{v} \end{array} \right) \lor \left(\begin{array}{l} \exists \ell, v', \vec{v}'', tl.\ hd = \textbf{some}\ l * \\ \qquad l \mapsto (v', \texttt{false}, tl) * \\ \qquad \vec{v} = v' :: \vec{v}'' * \Psi\ tl\ \vec{v}'' \end{array} \right)$$

Normalization

- Introduce quantifiers and modalities
  $\rightarrow$ Application step

Application

- Apply reflexivity
- Apply assumption
- Apply signature $\rightarrow$ Normalization step

$$(-\!\!*) \Longrightarrow (-\!\!*) \Longrightarrow (-\!\!*)$$

Proof search

$$(hd = \textbf{none} * \vec{v} = [])$$

$$\rightarrow^*$$

$$(hd = \textbf{none} * \vec{v} = [])$$

Normalization

- Introduce quantifiers and modalities
  $\rightarrow$ Application step

Application

- Apply reflexivity
- Apply assumption
- Apply signature $\rightarrow$ Normalization step

Proof search

$$(hd = \textbf{none} * \vec{v} = [])$$

$$\twoheadrightarrow$$

$$(hd = \textbf{none} * \vec{v} = [])$$

Normalization

- Introduce quantifiers and modalities
  $\rightarrow$ Application step

Application

- Apply reflexivity
- Apply assumption
- Apply signature $\rightarrow$ Normalization step

Proof search

$$
\begin{pmatrix}
\exists \ell, \surd, tl.hd = \mathbf{some}\ l \ast \\
\quad l \mapsto (\surd, \mathbf{true}, tl) \ast \\
\quad \Phi\ tl\ \vec{v}
\end{pmatrix} \vee
\begin{pmatrix}
\exists \ell, \surd, \vec{v}'', tl.\ hd = \mathbf{some}\ l \ast \\
\quad l \mapsto (\surd, \mathbf{false}, tl) \ast \\
\quad \vec{v} = \surd :: \vec{v}'' \ast \Phi\ tl\ \vec{v}''
\end{pmatrix}
$$

$$-\!\!\ast$$

$$
\begin{pmatrix}
\exists \ell, \surd, tl.hd = \mathbf{some}\ l \ast \\
\quad l \mapsto (\surd, \mathbf{true}, tl) \ast \\
\quad \Psi\ tl\ \vec{v}
\end{pmatrix} \vee
\begin{pmatrix}
\exists \ell, \surd, \vec{v}'', tl.\ hd = \mathbf{some}\ l \ast \\
\quad l \mapsto (\surd, \mathbf{false}, tl) \ast \\
\quad \vec{v} = \surd :: \vec{v}'' \ast \Psi\ tl\ \vec{v}''
\end{pmatrix}
$$

Normalization

- Introduce quantifiers and modalities
  $\rightarrow$ Application step

Application

- Apply reflexivity
- Apply assumption
- Apply signature $\rightarrow$ Normalization step

Proof search

$$\begin{pmatrix} \exists \ell, \sqrt{}, tl. hd = \mathbf{some}\ l * \\ \quad l \mapsto (\sqrt{}, \mathbf{true}, tl) * \\ \quad \Phi\ tl\ \vec{v} \end{pmatrix} \vee \begin{pmatrix} \exists \ell, \sqrt{}, \vec{v}'', tl.\ hd = \mathbf{some}\ l * \\ \quad l \mapsto (\sqrt{}, \mathbf{false}, tl) * \\ \quad \vec{v} = \sqrt{} :: \vec{v}'' * \Phi\ tl\ \vec{v}'' \end{pmatrix}$$

$$-\!\!*$$

$$\begin{pmatrix} \exists \ell, \sqrt{}, tl. hd = \mathbf{some}\ l * \\ \quad l \mapsto (\sqrt{}, \mathbf{true}, tl) * \\ \quad \Psi\ tl\ \vec{v} \end{pmatrix} \vee \begin{pmatrix} \exists \ell, \sqrt{}, \vec{v}'', tl.\ hd = \mathbf{some}\ l * \\ \quad l \mapsto (\sqrt{}, \mathbf{false}, tl) * \\ \quad \vec{v} = \sqrt{} :: \vec{v}'' * \Psi\ tl\ \vec{v}'' \end{pmatrix}$$

Normalization

- Introduce quantifiers and modalities
  $\rightarrow$ Application step

Application

- Apply reflexivity
- Apply assumption
- Apply signature $\rightarrow$ Normalization step

$$(-\!\!*) \Longrightarrow (-\!\!*) \Longrightarrow (-\!\!*)$$

Proof search

$$\Box\,(\forall hd\,\vec{v}.\,\Phi\;hd\,\vec{v} \twoheadrightarrow\!\!* \;\Psi\;hd\,\vec{v})$$
$$\vdash \Phi\;tl\,\vec{v} \twoheadrightarrow\!\!* \;\Psi\;tl\,\vec{v}$$

Normalization

- Introduce quantifiers and modalities
  $\rightarrow$ Application step

Application

- Apply reflexivity
- Apply assumption
- Apply signature $\rightarrow$ Normalization step

Proof search

$$\Box\,(\forall hd\,\vec{v}.\,\Phi\;hd\;\vec{v} \twoheadrightarrow \ast\;\Psi\;hd\;\vec{v})$$
$$\vdash \Phi\;tl\;\vec{v} \twoheadrightarrow \ast\;\Psi\;tl\;\vec{v}$$

Normalization

- Introduce quantifiers and modalities
  $\rightarrow$ Application step

Application

- Apply reflexivity
- Apply assumption
- Apply signature $\rightarrow$ Normalization step

Least fixpoint

Theorem (Least fixpoint)

Given a monotone function $\mathsf{F} \colon (A \to iProp) \to A \to iProp$*, called the pre fixpoint function, there exists the least fixpoint*

$$\mu\mathsf{F} \colon A \to iProp \triangleq \lambda \mathsf{F}\, x.\, \forall \Phi.\ \Box(\forall y.\, \mathsf{F}\, \Phi\, y \twoheadrightarrow \Phi\, y) \twoheadrightarrow \Phi\, x$$

such that

1. *The fixpoint equality holds*
$$\mu\mathsf{F}\, x \dashv\vdash \mathsf{F}\,(\mu\mathsf{F})\, x$$
2. *The iteration property holds*
$$\Box(\forall y.\, \mathsf{F}\, \Phi\, y \twoheadrightarrow \Phi\, y) \vdash \forall x.\, \mu\mathsf{F}\, x \twoheadrightarrow \Phi\, x$$

Elpi

- λProlog dialect (Dunchev, Guidi, Coen, and Tassi 2015)

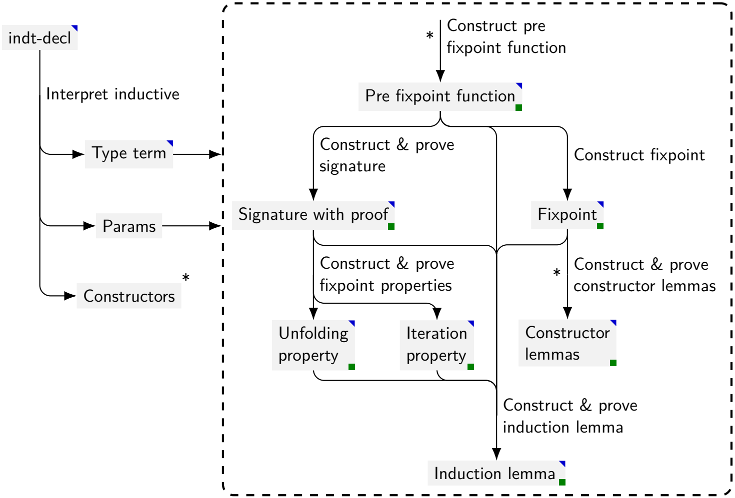
Elpi

- λProlog dialect (Dunchev, Guidi, Coen, and Tassi 2015)
- Coq meta-programming language (Tassi 2018)

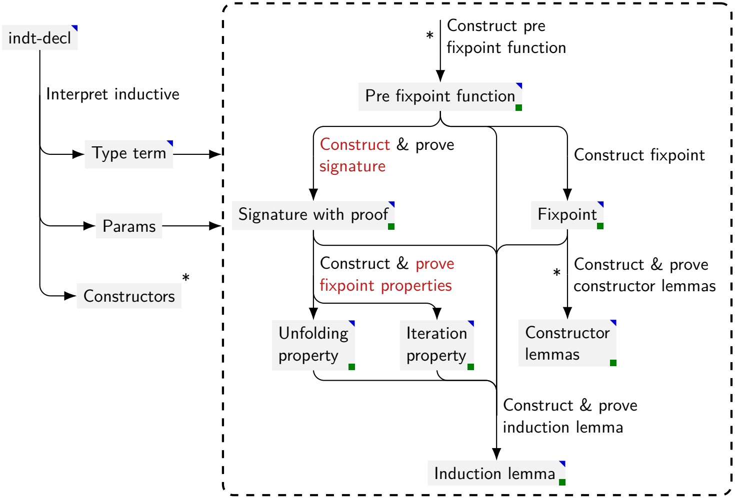
Elpi

- λProlog dialect (Dunchev, Guidi, Coen, and Tassi 2015)
- Coq meta-programming language (Tassi 2018)
- Derive (Tassi 2019)
- Hierarchy Builder (Cohen, Sakaguchi, and Tassi 2020)
- Trocq (Cohen, Crance, and Mahboubi 2024)

Outline of implementation



Outline of implementation



Coq-Elpi HOAS

```coq
val → list val → iProp
```

Coq-Elpi HOAS

```coq
val → list val → iProp
```

```coq
∀ _:val. ∀ _:list val. iProp
```

Coq-Elpi HOAS

```
val → list val → iProp
```
Coq

```
∀ _:val. ∀ _:list val. iProp
```
Coq

```
prod `_` (global (indt «val»))
      c0 \
         prod `_` (app [global (indt «list»),
                         global (indt «val»)])
              c1 \
                 app [global (indt «uPred»),
                      app [global (const «iResUR»),
                      global (const «Σ»)]]
```
Elpi

Generating terms

```
∀ _:val. ∀ _:list val. iProp
```

Coq

$$\square(\gtrdot \gtrdot (-\!\!*)) \Longrightarrow \gtrdot \gtrdot (-\!\!*)$$

```
pred type->signature i:term, i:term, o:term.
type->signature PreFixF Type Proper :-
    type->signature.aux Type P,
    coq.elaborate-skeleton
        {{ IProper (□> lp:P ==> lp:P) lp:PreFixF }}
        {{ Prop }} Proper ok.

pred type->signature.aux i:term, o:term.
type->signature (prod N T F) {{ .> lp:P }} :-
    pi x\ type->signature.aux (F x) P.
type->signature {{ iProp }} {{ bi_wand }}.
```

Elpi

Generating terms

```
∀ _:val. ∀ _:list val. iProp
```
Coq

$$\Box(\triangleright\triangleright(-\!\ast)) \Longrightarrow \triangleright\triangleright(-\!\ast)$$

```
pred type->signature i:term, i:term, o:term.
type->signature PreFixF Type Proper :-
    type->signature.aux Type P,
    coq.elaborate-skeleton
        {{ IProper (□> lp:P ==> lp:P) lp:PreFixF }}
        {{ Prop }} Proper ok.

pred type->signature.aux i:term, o:term.
type->signature (prod N T F) {{ .> lp:P }} :-
    pi x\ type->signature.aux (F x) P.
type->signature {{ iProp }} {{ bi_wand }}.
```
Elpi

Generating terms

```
∀ _:val. ∀ _:list val. iProp
```
Coq

$$\square(\triangleright\ \triangleright\ (-\!\!*)) \Longrightarrow \triangleright\ \triangleright\ (-\!\!*)$$

```
pred type->signature i:term, i:term, o:term.
type->signature PreFixF Type Proper :-
    type->signature.aux Type P,
    coq.elaborate-skeleton
        {{ IProper (□> lp:P ==> lp:P) lp:PreFixF }}
        {{ Prop }} Proper ok.

pred type->signature.aux i:term, o:term.
type->signature (prod N T F) {{ .> lp:P }} :-
    pi x\ type->signature.aux (F x) P.
type->signature {{ iProp }} {{ bi_wand }}.
```
Elpi

Generating terms

```
∀ _:val. ∀ _:list val. iProp
```

Coq

$$\square(\gtrdot \gtrdot (-\!*)) \Longrightarrow \gtrdot \gtrdot (-\!*)$$

```
pred type->signature i:term, i:term, o:term.
type->signature PreFixF Type Proper :-
    type->signature.aux Type P,
    coq.elaborate-skeleton
        {{ IProper (□> lp:P ==> lp:P) lp:PreFixF }}
        {{ Prop }} Proper ok.

pred type->signature.aux i:term, o:term.
type->signature (prod N T F) {{ .> lp:P }} :-
    pi x\ type->signature.aux (F x) P.
type->signature {{ iProp }} {{ bi_wand }}.
```

Elpi

Generating terms

```
∀ _:val. ∀ _:list val. iProp
```

Coq

$$\square(\gg \gg (-\!\!*)) \Longrightarrow \gg \gg (-\!\!*)$$

```
pred type->signature i:term, i:term, o:term.
type->signature PreFixF Type Proper :-
    type->signature.aux Type P,
    coq.elaborate-skeleton
        {{ IProper (□> lp:P ==> lp:P) lp:PreFixF }}
        {{ Prop }} Proper ok.

pred type->signature.aux i:term, o:term.
type->signature (prod N T F) {{ .> lp:P }} :-
    pi x\ type->signature.aux (F x) P.
type->signature {{ iProp }} {{ bi_wand }}.
```

Elpi

Generating terms

```
∀ _:val. ∀ _:list val. iProp
```
Coq

$$\square(\gtrdot\ \gtrdot\ (\mathrel{-\!\!*})) \Longrightarrow \gtrdot\ \gtrdot\ (\mathrel{-\!\!*})$$

```
pred type->signature i:term, i:term, o:term.
type->signature PreFixF Type Proper :-
    type->signature.aux Type P,
    coq.elaborate-skeleton
        {{ IProper (□> lp:P ==> lp:P) lp:PreFixF }}
        {{ Prop }} Proper ok.

pred type->signature.aux i:term, o:term.
type->signature (prod N T F) {{ .> lp:P }} :-
    pi x\ type->signature.aux (F x) P.
type->signature {{ iProp }} {{ bi_wand }}.
```
Elpi

Generating terms

```coq
∀ _:val. ∀ _:list val. iProp
```
Coq

$$\Box(\gtrdot\ \gtrdot\ (-\!\!*)) \implies \gtrdot\ \gtrdot\ (-\!\!*)$$

```elpi
pred type->signature i:term, i:term, o:term.
type->signature PreFixF Type Proper :-
    type->signature.aux Type P,
    coq.elaborate-skeleton
        {{ IProper (□> lp:P ==> lp:P) lp:PreFixF }}
        {{ Prop }} Proper ok.

pred type->signature.aux i:term, o:term.
type->signature (prod N T F) {{ .> lp:P }} :-
    pi x\ type->signature.aux (F x) P.
type->signature {{ iProp }} {{ bi_wand }}.
```
Elpi

Generating proofs

- Proofs are also just terms

Generating proofs

- Proofs are also just terms
- Other Elpi projects just generate proof terms using roughly the previous method

Generating proofs

- Proofs are also just terms
- Other Elpi projects just generate proof terms using roughly the previous method
- We reuse the IPM lemmas written for its tactics

Generating proofs

- Proofs are also just terms
- Other Elpi projects just generate proof terms using roughly the previous method
- We reuse the IPM lemmas written for its tactics
- We develop a strategy for modular proof term generators which employ the Coq API

Generating proofs

- Proofs are also just terms
- Other Elpi projects just generate proof terms using roughly the previous method
- We reuse the IPM lemmas written for its tactics
- We develop a strategy for modular proof term generators which employ the Coq API
- These modular proof term generators are also repackaged into a new set of IPM tactics, e.g. `eiIntros` (Elpi Iris Intros).

Composing proof generators

Coq

```
hd : val
vs : list val
----------------------------------------
is_MLL_pre is_MLL hd vs ⊣⊢ is_MLL hd vs
```

Elpi

```
pred mk-unfold.proof i:int, i:term, i:term, i:hole.
mk-unfold.proof Ps Unfold1 Unfold2 H :-
  do-iStartProof H (ihole N H'), !,
  do-iAndSplit H' H1 H2,
  std.map {std.iota Ps} (x\r\ r = {{ _ }}) Holes1, !,
  do-iApplyLem (app [Unfold1 | Holes1]) (ihole N H1) [] [], !,
  std.map {std.iota Ps} (x\r\ r = {{ _ }}) Holes2, !,
  do-iApplyLem (app [Unfold2 | Holes2]) (ihole N H2) [] [].
```

Demo

Conclusion

- Created a system for defining and using inductive predicates in the IPM

Conclusion

- Created a system for defining and using inductive predicates in the IPM
- Posed a strategy for defining modular tactics in Elpi

Conclusion

- Created a system for defining and using inductive predicates in the IPM
- Posed a strategy for defining modular tactics in Elpi
- Posed a syntactic proof search algorithm for finding a monotonicity proof of a pre fixpoint function

Conclusion

- Created a system for defining and using inductive predicates in the IPM
- Posed a strategy for defining modular tactics in Elpi
- Posed a syntactic proof search algorithm for finding a monotonicity proof of a pre fixpoint function
- Evaluated Elpi as a meta-programming language for the IPM

Questions

Future work

- Non-expansive predicates
- Other fixpoint predicates
- Nested inductive predicates
- Mutual inductive predicates

Holes in proofs

Elpi

```
kind hole type.
type hole term -> term -> hole.

pred do-iAndSplit i:hole, o:hole, o:hole.
do-iAndSplit (hole Type Proof) (hole LType LProof)
             (hole RType RProof) :-
  @no-tc! => coq.elaborate-skeleton
               {{ tac_and_split _ _ _ _ _ _ _ }}
               Type Proof ok,
  Proof = {{ tac_and_split _ _ _ _
                         lp:FromAnd lp:LProof lp:RProof }},
  coq.ltac.collect-goals FromAnd [G1] _,
  open tc_solve G1 [],
  coq.typecheck LProof LType ok,
  coq.typecheck RProof RType ok.
```

Monotonicity as a signature

Definition (Proper element of a relation (Sozeau 2009))

Given a relation R: *iRel* A and an element $x \in A$, x is a proper element of R if $R\,x\,x$.

Monotonicity as a signature

Definition (Proper element of a relation (Sozeau 2009))

Given a relation R: *iRel* A and an element $x \in A$, x is a proper element of R if $R\,x\,x$.

Definition (Respectful relation)

$$R \Longrightarrow R' \triangleq \lambda f, g.\ \forall x, y.\ R\,x\,y \mathrel{-\!\!*} R'\,(f\,x)\,(g\,y)$$

Definition (Pointwise relation)

$$\gtrdot R \triangleq \lambda f, g.\ \forall x.\ R\,(f\,x)\,(g\,x)$$

Definition (Persistent relation)

$$\blacksquare R \triangleq \lambda x, y.\ \Box(R\,x\,y)$$

Monotonicity as a signature

Definition (Proper element of a relation (Sozeau 2009))

Given a relation R: *iRel* A and an element $x \in A$, x is a proper element of R if $R\,x\,x$.

Definition (Respectful relation)

$$R \Longrightarrow R' \triangleq \lambda f, g.\, \forall x, y.\, R\,x\,y \twoheadrightarrow\!\!* R'\,(f\,x)\,(g\,y)$$

Definition (Pointwise relation)

$$\gtrdot R \triangleq \lambda f, g.\, \forall x.\, R\,(f\,x)\,(g\,x)$$

Definition (Persistent relation)

$$\blacksquare R \triangleq \lambda x, y.\, \Box(R\,x\,y)$$

$$\mathsf{isMLL}_\mathsf{F} : (\mathit{Val} \to \mathit{List}\,\mathit{Val} \to \mathit{iProp}) \to \mathit{Val} \to \mathit{List}\,\mathit{Val} \to \mathit{iProp}$$

$$\Box\,(\forall hd\,\vec{v}.\, \Phi\,hd\,\vec{v} \mathrel{-\!\!*} \Psi\,hd\,\vec{v}) \mathrel{-\!\!*} \left(\forall hd\,\vec{v}.\quad \begin{array}{l} \mathsf{isMLL}_\mathsf{F}\,\Phi\,hd\,\vec{v} \mathrel{-\!\!*} \\ \mathsf{isMLL}_\mathsf{F}\,\Psi\,hd\,\vec{v} \end{array}\right)$$

Monotonicity as a signature

Definition (Proper element of a relation (Sozeau 2009))

Given a relation R: $iRel\ A$ and an element $x \in A$, x is a proper element of R if $R\,x\,x$.

Definition (Respectful relation)

$$R \Longrightarrow R' \triangleq \lambda f, g.\ \forall x, y.\ R\,x\,y \mathrel{-\!\!*} R'\,(f\,x)\,(g\,y)$$

Definition (Pointwise relation)

$$\gtrdot R \triangleq \lambda f, g.\ \forall x.\ R\,(f\,x)\,(g\,x)$$

Definition (Persistent relation)

$$\Box R \triangleq \lambda x, y.\ \Box(R\,x\,y)$$

$$\mathrm{isMLL}_{\mathrm{F}} : (Val \to List\,Val \to iProp) \to Val \to List\,Val \to iProp$$

$$\Box\,(\forall hd\,\vec{v}.\ \Phi\ hd\ \vec{v} \mathrel{-\!\!*} \Psi\ hd\ \vec{v}) \mathrel{-\!\!*} \left(\forall hd\,\vec{v}.\quad \begin{array}{l} \mathrm{isMLL}_{\mathrm{F}}\ \Phi\ hd\ \vec{v} \mathrel{-\!\!*} \\ \mathrm{isMLL}_{\mathrm{F}}\ \Psi\ hd\ \vec{v} \end{array} \right)$$

$$\Box(\gtrdot \gtrdot (\mathrel{-\!\!*})) \Longrightarrow \gtrdot \gtrdot (\mathrel{-\!\!*})$$