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
- Specification happens in separation logic
- We make use of embeddings of separation logic in a proof assistant
- Iris (Jung, Krebbers, Jourdan, Bizjak, Birkedal, and Dreyer 2018) & Coq (Huet, Kahn, and Paulin-Mohring 2002)

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

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

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

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

Imperative program

Functional program

Separation logic with Hoare triples

Representation predicate

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

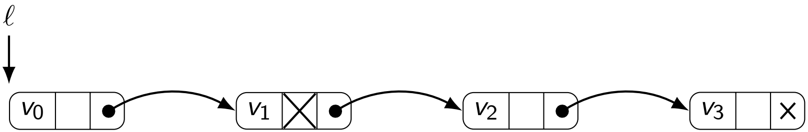
Imperative program

Functional program

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

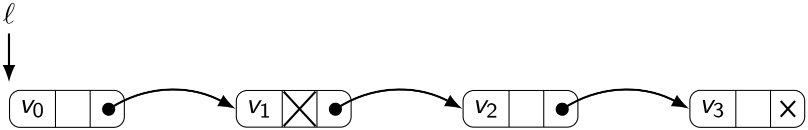
Representation predicates

Representation predicates



Harris (2001)

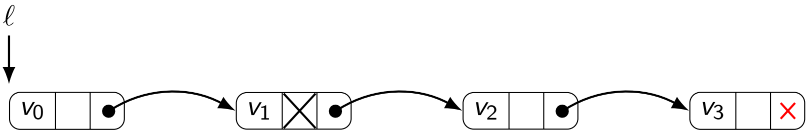
Representation predicates



Harris (2001)

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

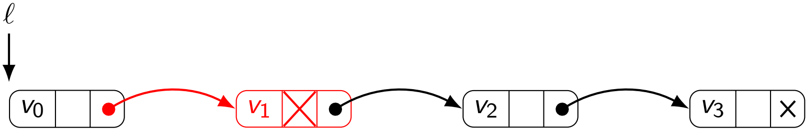
Representation predicates



Harris (2001)

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

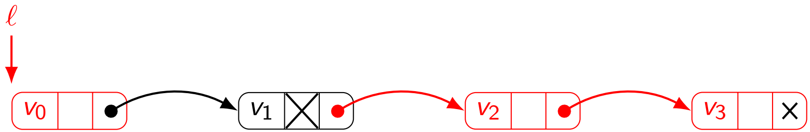
Representation predicates



Harris (2001)

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

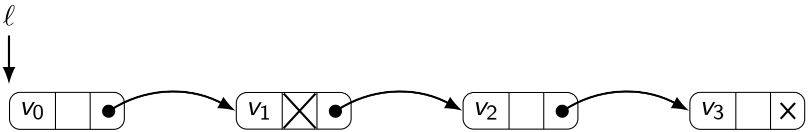
Representation predicates



Harris (2001)

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

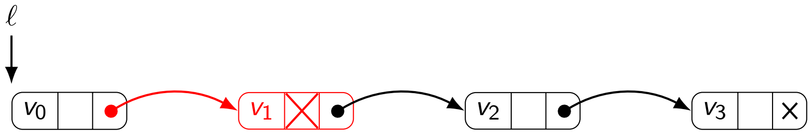
Representation predicates



Harris (2001)

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

Representation predicates



Harris (2001)

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

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

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).
```

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

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).
```

- 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
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
- 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

Contributions

Contributions

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

Contributions

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

Contributions

- 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

Contributions

- 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

Monotone pre fixpoint function

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

Monotone pre fixpoint function

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

Monotone pre fixpoint function

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

Definition (Monotone predicate)

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

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

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 $R\,x\,x$.

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)

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

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)

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

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

$$\square\,(\forall hd\,\vec{v}.\, \Phi\,hd\,\vec{v} \twoheadrightarrow \Psi\,hd\,\vec{v}) \twoheadrightarrow \left(\forall hd\,\vec{v}.\quad \begin{array}{l} \mathrm{isMLL}_{\mathrm{F}}\,\Phi\,hd\,\vec{v} \twoheadrightarrow \\ \mathrm{isMLL}_{\mathrm{F}}\,\Psi\,hd\,\vec{v} \end{array} \right)$$

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)

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

$$\mathrm{isMLL_F} : (Val \to List\,Val \to iProp) \to Val \to List\,Val \to iProp$$

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

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

Monotone signatures

Connective	Type	Signature
$*$	$iProp \to iProp \to iProp$	$(-\!\!*) \Longrightarrow (-\!\!*) \Longrightarrow (-\!\!*)$
$\vee$	$iProp \to iProp \to iProp$	$(-\!\!*) \Longrightarrow (-\!\!*) \Longrightarrow (-\!\!*)$
$-\!\!*$	$iProp \to iProp \to iProp$	$\mathrm{flip}(-\!\!*) \Longrightarrow (-\!\!*) \Longrightarrow (-\!\!*)$
$\exists$	$(A \to iProp) \to iProp$	$\gtrdot(-\!\!*) \Longrightarrow (-\!\!*)$

Signature	Semantics
$(-\!\!*) \Longrightarrow (-\!\!*) \Longrightarrow (-\!\!*)$	$\forall P, P'.\,(P -\!\!* P') -\!\!* \forall Q, Q'.\,(Q -\!\!* Q') -\!\!* (P * Q) -\!\!* (P' * Q')$
$(\gtrdot(-\!\!*) \Longrightarrow (-\!\!*))$	$\forall \Phi, \Psi.\,(\forall x.\,\Phi\,x -\!\!* \Psi\,x) -\!\!* (\exists x.\,\Phi\,x) -\!\!* (\exists x.\,\Psi\,x)$

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
  $\to$ Application step

Application

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

Proof search

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

$$-\!*$$

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

Normalization

- Introduce quantifiers and modalities
  $\rightarrow$ Application step

Application

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

Proof search

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

$-\!*$

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

Normalization

- Introduce quantifiers and modalities
  $\to$ Application step

Application

- Apply reflexivity
- Apply assumption
- Apply signature $\to$ 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

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

$$\twoheadrightarrow*$$

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

Normalization

- Introduce quantifiers and modalities
  $\rightarrow$ Application step

Application

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

Proof search

$$
\begin{aligned}
&\left(\begin{aligned}\exists \ell, \surd, tl.hd = \mathbf{some}\ l \ast \\ l \mapsto (\surd, \mathbf{true}, tl) \ast \\ \Phi\ tl\ \vec{v}\end{aligned}\right) \vee \\
&\left(\begin{aligned}\exists \ell, \surd, \vec{v}'', tl.\ hd = \mathbf{some}\ l \ast \\ l \mapsto (\surd, \mathbf{false}, tl) \ast \\ \vec{v} = \surd :: \vec{v}'' \ast \Phi\ tl\ \vec{v}''\end{aligned}\right) \\
&\twoheadrightarrow\!\!\ast \\
&\left(\begin{aligned}\exists \ell, \surd, tl.hd = \mathbf{some}\ l \ast \\ l \mapsto (\surd, \mathbf{true}, tl) \ast \\ \Psi\ tl\ \vec{v}\end{aligned}\right) \vee \\
&\left(\begin{aligned}\exists \ell, \surd, \vec{v}'', tl.\ hd = \mathbf{some}\ l \ast \\ l \mapsto (\surd, \mathbf{false}, tl) \ast \\ \vec{v} = \surd :: \vec{v}'' \ast \Psi\ tl\ \vec{v}''\end{aligned}\right)
\end{aligned}
$$

Normalization

- Introduce quantifiers and modalities
  $\rightarrow$ Application step

Application

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

Proof search

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

$-\!\!*$

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

Normalization

- Introduce quantifiers and modalities
  $\rightarrow$ Application step

Application

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

Proof search

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

Normalization

- Introduce quantifiers and modalities
  $\rightarrow$ Application step

Application

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

Proof search

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

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*

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

such that

1. *The fixpoint equality holds*

$$\mu\mathsf{F}\, x \dashv\vdash \mathsf{F}\,(\mu\mathsf{F})\, x$$

2. *The iteration property holds*

$$\Box(\forall y.\, \mathsf{F}\, \Phi\, y \mathrel{-\!\!*} \Phi\, y) \vdash \forall x.\, \mu\mathsf{F}\, x \mathrel{-\!\!*} \Phi\, x$$

Elpi

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

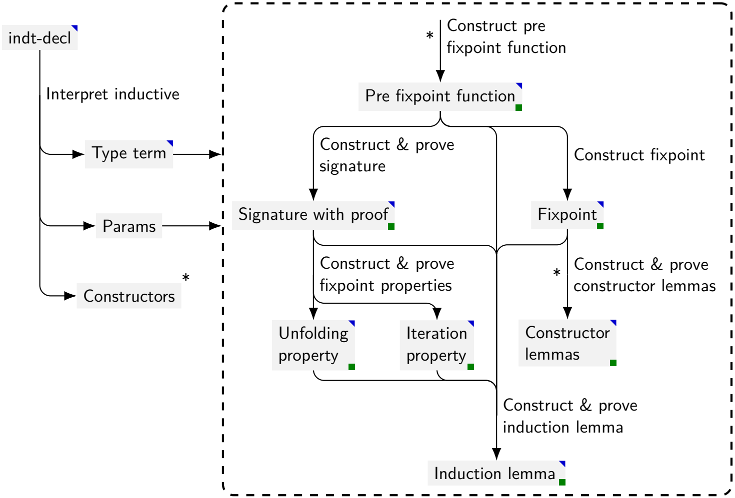
Elpi

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

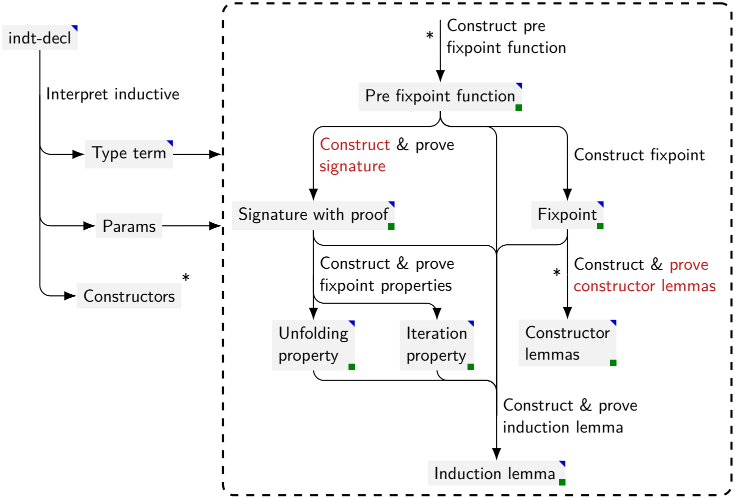
Elpi

- 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

```
val → list val → iProp
```

Coq

Coq-Elpi HOAS

```coq
val → list val → iProp
```

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

Coq-Elpi HOAS

```coq
val → list val → iProp
```

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

```elpi
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»)]]
```

Generating terms

Elpi

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

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

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 want to 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 want to reuse the IPM lemmas written for its tactics
- They depend on typeclass search to find the next step to prove

Generating proofs

- Proofs are also just terms
- Other Elpi projects just generate proof terms using roughly the previous method
- We want to reuse the IPM lemmas written for its tactics
- They depend on typeclass search to find the next step to prove
- We develop a strategy for modular proof term generators which employ the Coq API to gives types to any holes.

Generating proofs

- Proofs are also just terms
- Other Elpi projects just generate proof terms using roughly the previous method
- We want to reuse the IPM lemmas written for its tactics
- They depend on typeclass search to find the next step to prove
- We develop a strategy for modular proof term generators which employ the Coq API to gives types to any holes.
- These modular proof term generators are also repackaged into a new set of IPM tactics, e.g. `eiIntros` (Elpi Iris Intros).

Holes in proofs

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

Elpi

```elpi
pred do-iLeft i:hole, o:hole.
do-iLeft (hole Type Proof) (hole LeftType LeftProof) :-
    @no-tc! => coq.elaborate-skeleton
                 {{ tac_or_l _ _ _ _ _ _ }}
                 Type Proof ok,
    Proof = {{ tac_or_l _ _ _ _ lp:FromOr lp:LeftProof }},
    coq.ltac.collect-goals FromOr [G1] _,
    open tc_solve G1 [],
    coq.typecheck LeftProof LeftType ok.
```

Elpi

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

Evaluation of Elpi

Conclusion

- Proof search algorithm for monotonicity of pre fixpoint functions

Conclusion

- Proof search algorithm for monotonicity of pre fixpoint functions
- Elpi implementation

Future work

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