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 et al. 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

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

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

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

Imperative program

Functional program

Separation logic with Hoare triples

Representation predicate

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

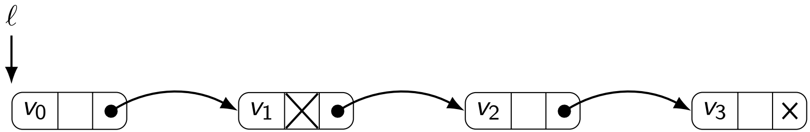
Imperative program

Functional program

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

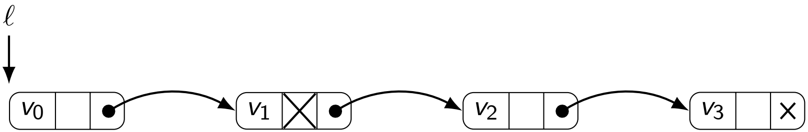
Representation predicates

Representation predicates



Harris (2001)

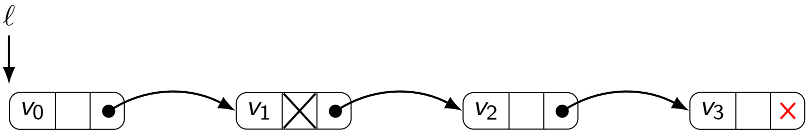
Representation predicates



Harris (2001)

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

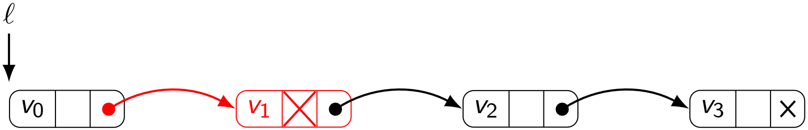
Representation predicates



Harris (2001)

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

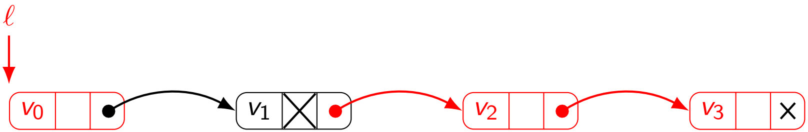
Representation predicates



Harris (2001)

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

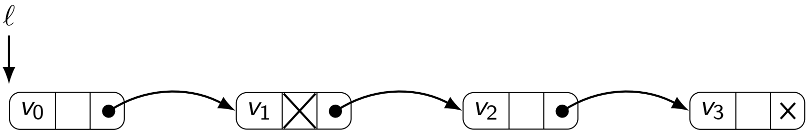
Representation predicates



Harris (2001)

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

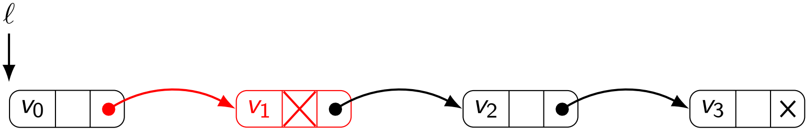
Representation predicates



Harris (2001)

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

Representation predicates



Harris (2001)

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

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

```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
- 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

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

Monotone pre fixpoint function

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

Monotone pre fixpoint function

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

Definition (Monotone predicate)

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

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

Monotone signatures

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

Monotone signatures

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

Definition (Respectful relation)

The *respectful relation* $R \Longrightarrow R' : iRel\ (A \to B)$ of two relations $R : iRel\ A$, $R' : iRel\ B$ is defined as

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

Semantics of a signature

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 $R\,x\,x$.

Semantics of a signature

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 $R\,x\,x$.

Signature	Semantics
$(-\!*) \Longrightarrow (-\!*) \Longrightarrow (-\!*)$	$\forall P, P'.\,(P -\!* P') -\!* \forall Q, Q'.\,(Q -\!* Q') -\!* (P * Q) -\!* (P' * Q')$

Semantics of a signature

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 $R\,x\,x$.

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

Monotonicity as a signature

Proof search

Elpi

Demo

Conclusion

Future work