Chapter 3

Fixpoints for representation predicates

In this chapter, we show how non-structurally recursive representation predicates can be defined using least fixpoints. In section 3.1, we explain why it is difficult to define non-structurally recursive predicates and generally explain the approach that is taken. Next, in section 3.2, we show the way least fixpoints are defined in Iris. The least fixpoint in Iris was never formalized in a paper, it has only been defined in Coq [Kre+24]. Lastly, in section 3.3, we explain the improvements we made to the approach of Iris in order for the process to be automated.

3.1 Problem statement

To define a recursive predicate, we have to prove it actually exists. One way of defining recursive predicates is by structural recursion. Thus, every recursive call in the predicate has to be on a structurally smaller part of the arguments.

The candidate argument for structural recursion in isMLL would be the list of values used to represent the MLL. However, this does not work given the second case of the recursion.

$$\mathsf{isMLL}\, hd\, \vec{v} = \cdots \vee (\exists \ell, v', tl.\, hd = \mathtt{\textbf{some}}\, \ell * \ell fmapsto(v', \mathtt{\textbf{true}}, tl) * \mathsf{isMLL}\, tl\, \vec{v}) \vee \cdots$$

Here, the list of values is passed straight onto the recursive call to isMLL. Thus, it is not structurally recursive.

We need another approach to define non-structurally recursive predicates such as these. Iris has several approaches to resolve this issue, as is discussed in chapter 7thesis.pdf. The approach we use as the basis of `eiInd` is the least fixpoint, inspired by the Knaster-Tarski fixpoint theorem [Tar55]. Given a monotone function on predicates, the least fixpoint of this function exists. We can now choose a function such that the fixpoint corresponds to the recursive predicate we wanted to design. This procedure is explained thoroughly in the next section, section 3.2.

3.2 Least fixpoint in Iris

To define the least fixpoint in Iris, the first step is to have a monotone function.

Definition 3.1: Monotone function

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

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

In other words, F is monotone in its first argument.

This definition of monotone follows the definition of monotone in other fields, with one exception. The assumption has an additional restriction, it has to be persistent. The persistence is necessary since F could use its monotone argument multiple times.

Example 3.2

Take the following function.

$$\begin{aligned}\mathsf{F}\, \Phi\, v \triangleq{} & (v = \textbf{none}) \vee \\ & (\exists \ell_1, \ell_2, v_1, v_2.\, v = \textbf{some}(\ell_1, \ell_2) * \ell_1 \mapsto v_1 * \ell_2 \mapsto v_2 * \Phi\, v_1 * \Phi\, v_2)\end{aligned}$$

This is the function for binary trees. The value v is either empty, and we have an empty tree. Or v contains two locations, for the two branches of the tree. Each location points to a value, and Φ holds for both of these values. The fixpoint, as is discussed in theorem 3.3, of this function holds for a value containing a binary tree. However, before we can take the fixpoint we have to prove it is monotone.

$$\square(\forall w.\, \Phi\, w \mathrel{-\!\!*} \Psi\, w) \vdash \forall v.\, \mathsf{F}\, \Phi\, v \mathrel{-\!\!*} \mathsf{F}\, \Psi\, v$$

Proof. We start by introducing v and the wand.

$$\square(\forall w.\, \Phi\, w \mathrel{-\!\!*} \Psi\, w) * \mathsf{F}\, \Phi\, v \vdash \mathsf{F}\, \Psi\, v$$

We now unfold the definition of F and eliminate and introduce the disjunction, resulting in two statements to prove.

$$\square(\forall w.\, \Phi\, w \mathrel{-\!\!*} \Psi\, w) * v = \textbf{none} \vdash v = \textbf{none}$$

$$\begin{aligned}&\square(\forall w.\, \Phi\, w \mathrel{-\!\!*} \Psi\, w) * \left(\exists \ell_1, \ell_2, v_1, v_2.\, \begin{array}{l} v = \textbf{some}(\ell_1, \ell_2) * \ell_1 \mapsto v_1 * \\ \ell_2 \mapsto v_2 * \Phi\, v_1 * \Phi\, v_2 \end{array}\right) \vdash \\ &\left(\exists \ell_1, \ell_2, v_1, v_2.\, \begin{array}{l} v = \textbf{some}(\ell_1, \ell_2) * \ell_1 \mapsto v_1 * \\ \ell_2 \mapsto v_2 * \Psi\, v_1 * \Psi\, v_2 \end{array}\right)\end{aligned}$$

The first statement holds directly. For the second statement, we eliminate the existentials in the assumption and use the created variables to introduce the existentials in the conclusion.

$$\square(\forall w.\, \Phi\, w \mathrel{-\!\!*} \Psi\, w) * \begin{array}{l} v = \textbf{some}(\ell_1, \ell_2) * \\ \ell_1 \mapsto v_1 * \ell_2 \mapsto v_2 * \\ \Phi\, v_1 * \Phi\, v_2 \end{array} \vdash \begin{array}{l} v = \textbf{some}(\ell_1, \ell_2) * \\ \ell_1 \mapsto v_1 * \ell_2 \mapsto v_2 * \\ \Psi\, v_1 * \Psi\, v_2 \end{array}$$

Any sub propositions that occur both on the left and right-hand side are canceled out using $*$-MONO.

$$\square(\forall w.\, \Phi\, w \mathrel{-\!\!*} \Psi\, w) * \Phi\, v_1 * \Phi\, v_2 \vdash \Psi\, v_1 * \Psi\, v_2$$

We want to split the conclusion and premise in two, such that we get the following statements, with $i \in \{1, 2\}$.

$$\square(\forall w.\, \Phi\, w \mathrel{-\!\!*} \Psi\, w) * \Phi\, v_i \vdash \Psi\, v_i$$

To achieve this split, we duplicate the persistent premise and then split using $*$-MONO again. Both these statements hold trivially. □

In the previous proof, it was essential that the premise of monotonicity is persistent. This occurs any time we have a data structure with more than one branch.

Now that we have a definition of a function, we can prove that the least fixpoint of a monotone function always exists.

Theorem 3.3: Least fixpoint

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

1. The fixpoint equality holds

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

2. The iteration property holds

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

Proof. Given a monotone function $\mathsf{F} \colon (A \to \mathit{iProp}) \to A \to \mathit{iProp}$ we define $\mu\mathsf{F}$ as

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

We now prove the two properties of the least fixpoint

1. The right to left direction follows from monotonicity of F. The left to right direction follows easily from monotonicity of F and the right to left direction.
2. This follows directly from unfolding the definition of $\mu\mathsf{F}$. □

The first property of theorem 3.3, fixpoint equality, defines that the least fixpoint is a fixpoint. The second property of theorem 3.3, iteration, ensures that this fixpoint is the least of the possible fixpoints. The iteration property is a simpler version of the induction principle. The induction hypothesis during iteration is simpler. It only ensures that Φ holds under F. Full induction requires that we also know that the fixpoint holds under F in the induction hypothesis.

Lemma 3.4: Induction principle

Given a monotone predicate $\mathsf{F}\colon (A \to \mathit{iProp}) \to (A \to \mathit{iProp})$, it holds that

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

Proof. The induction principle for a Ψ holds by the iteration property with $\mathit{\Phi} x = \Psi\, x \wedge \mu\mathsf{F}\, x$ □

This lemma follows from monotonicity and the least fixpoint properties.
We can now use the above steps to define isMLL.

Example 3.5: Iris least fixpoint of isMLL

We want to create the least fixpoint such that it has the following inductive property.

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

The first step is creating the pre fixpoint function. We accomplish this by adding an argument to isMLL and then transforming it into a function. Next, we substitute any recursive calls to isMLL with this argument.

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

This has created a function, $\mathsf{isMLL}_{\mathsf{F}}$. The function applies the predicate, $\mathit{\Phi}$, on the tail of any possible MLL, while ensuring the head is part of an MLL. Next, we want to prove that $\mathsf{isMLL}_{\mathsf{F}}$ is monotone. However, $\mathsf{isMLL}_{\mathsf{F}}$ has the following type.

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

But, definition 3.1 only works for functions of type

$$\mathsf{F}\colon (A \to \mathit{iProp}) \to A \to \mathit{iProp}$$

This is solved by uncurrying $\mathsf{isMLL}_{\mathsf{F}}$

$$\mathsf{isMLL}'_{\mathsf{F}}\, \mathit{\Phi}\, (hd, \vec{v}) \triangleq \mathsf{isMLL}_{\mathsf{F}}\, \mathit{\Phi}\, hd\, \vec{v}$$

The function $\mathsf{isMLL}'_{\mathsf{F}}$ now has the type

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

And we can prove $\mathsf{isMLL}_{\mathsf{F}}$ is monotone.

$$\begin{gathered}\Box(\forall (hd, \vec{v}).\, \mathit{\Phi}\, (hd, \vec{v}) \mathrel{-\!\!*} \Psi\, (hd, \vec{v})) \\ \vdash \forall (hd, \vec{v}).\, \mathsf{isMLL}'_{\mathsf{F}}\, \mathit{\Phi}\, (hd, \vec{v}) \mathrel{-\!\!*} \mathsf{isMLL}'_{\mathsf{F}}\, \Psi\, (hd, \vec{v})\end{gathered}$$

Proof. We use a similar proof as in example 3.2. It involves more steps as we have more branches, but the same ideas apply. □

Given that $\mathsf{isMLL}'_{\mathsf{F}}$ is monotone, we now know from theorem 3.3 that the least fixpoint exists of $\mathsf{isMLL}'_{\mathsf{F}}$. By uncurrying we can create the final definition of isMLL.

$$\mathsf{isMLL}\, hd\, \vec{v} \triangleq \mu(\mathsf{isMLL}'_{\mathsf{F}})\,(hd, \vec{v})$$

This definition of isMLL has the inductive property as described in section 2.5thesis.pdf. That property is the fixpoint equality. After expanding any currying, we get the below induction principle for isMLL from lemma 3.4.

$$\begin{aligned}&\Box(\forall hd, \vec{v}.\, \mathsf{isMLL}_{\mathsf{F}}\,(\lambda hd', \vec{v}'.\, \Phi\, hd'\, \vec{v}' \wedge \mathsf{isMLL}\, hd'\, \vec{v}')\, hd\, \vec{v} \mathrel{-\!\!*} \Phi\, hd\, \vec{v})\\ &\mathrel{-\!\!*} \forall hd, \vec{v}.\, \mathsf{isMLL}\, hd\, \vec{v} \mathrel{-\!\!*} \Phi\, hd\, \vec{v}\end{aligned}$$

The induction principle from section 2.5thesis.pdf is also derivable from lemma 3.4. The three cases of the induction principle follow from the disjunctions in $\mathsf{isMLL}_{\mathsf{F}}$.

3.3 Syntactic monotone proof search

As we discussed in chapter 1thesis.pdf, the goal of this thesis is to show how to automate the definition of representation predicates from inductive definitions. The major hurdle in this process can be seen in example 3.5. Proving a function monotone. In this section, we show how a monotonicity proof can be found by using a syntactic proof search.

We base our strategy on the work by Sozeau [Soz09]. They create a system for rewriting expressions in goals in Coq under generalized relations, instead of just equality. Many definitions are equivalent, but we do them in the embedded separation logic instead of the logic of Coq. The proof search itself is not based on the generalized rewriting of Sozeau.

We take the following strategy. We prove the monotonicity of all the connectives once. Now, we prove the monotonicity of the function by making use of the monotonicity of the connectives with which it is built.

Monotone connectives We do not want to uncurry every connective when using its monotonicity. Thus, we take a different approach to what is monotone than Iris in the previous section. For every connective, we give a signature telling us how it is monotone for its arguments. We show a few of these signatures below.

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

We make use of the Haskell prefix notation, $(\mathrel{-\!\!*})$, to turn an infix operator into a prefix function. The signature of a connective defines the requirements for monotonicity a connective has. The signatures are based on building relations, which we can apply on the connectives.

Definition 3.6: Relation in *iProp*

A relation in separation logic on type A is defined as

$$\mathit{iRel}\ A \triangleq A \to A \to \mathit{iProp}$$

The combinators used to build signatures now build relations.

Definition 3.7: Respectful relation

The respectful relation $R \Longrightarrow R' \colon \mathit{iRel}\ (A \to B)$ of two relations $R \colon \mathit{iRel}\ A$, $R' \colon \mathit{iRel}\ B$ is defined as

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

Definition 3.8: Flipped relation

The flipped relation $\mathsf{flip}\, R \colon \mathit{iRel}\ A$ of a relation $R \colon \mathit{iRel}\ A$ is defined as

$$\mathsf{flip}\, R \triangleq \lambda x, y.\, R\, y\, x$$

Given a signature we can define when a connective has a signature.

Definition 3.9: Proper element of a relation

Given a relation $R \colon \mathit{iRel}\ A$ and an element $x \in A$, x is a proper element of R if $R\, x\, x$.

We define how a connective is monotone by the signature it is a proper element of. The proofs that the connectives are the proper elements of their signature are fairly trivial, but we will highlight the existential qualifier.

Recall the existential quantifieres signature, $((=) \Longrightarrow (\mathrel{-\!\!*})) \Longrightarrow (\mathrel{-\!\!*})$. We can unfold the definitions in the signature and fill in the existential quantification to get the following statement,

$$\forall \varPhi, \Psi.\, (\forall x, y.\, x = y \mathrel{-\!\!*} \varPhi\, x \mathrel{-\!\!*} \Psi\, y) \mathrel{-\!\!*} (\exists x.\, \varPhi\, x) \mathrel{-\!\!*} (\exists x.\, \Psi\, x)$$

This statement can be easily simplified by substituting y for x in the first relation.

$$\forall \varPhi, \Psi.\, (\forall x.\, \varPhi\, x \mathrel{-\!\!*} \Psi\, x) \mathrel{-\!\!*} (\exists x.\, \varPhi\, x) \mathrel{-\!\!*} (\exists x.\, \Psi\, x)$$

We create a new combinator for signatures, the pointwise relation, to include the above simplification in signatures.

Definition 3.10: Pointwise relation

The pointwise relation $\gtrdot R$ is a special case of a respectful relation defined as

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

The new signature for the existential quantification becomes

$$> (\mathrel{-\!\!*}) \Longrightarrow (\mathrel{-\!\!*})$$

Monotone functions To create a monotone function for the least fixpoint we need to be able to restate definition 3.1 in terms of the proper element of a signature. We already have most of the combinators needed, but we are missing a way to mark a relation as persistent.

Definition 3.11: Persistent relation

The persistent relation $\square R\colon \mathit{iRel}\ A$ for a relation $R\colon \mathit{iRel}\ A$ is defined as

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

Thus, we can create the following signature for definition 3.1.

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

Filling in an F as the proper element, we get the following statement.

$$\square(\forall y.\, \mathit{\Phi}\, y \mathrel{-\!\!*} \Psi\, y) \mathrel{-\!\!*} \forall x.\, \mathsf{F}\, \mathit{\Phi}\, x \mathrel{-\!\!*} \mathsf{F}\, \Psi\, x$$

Which is definition 3.1 but using only wands, instead of entailments. We use the same structure for the signature of $\mathsf{isMLL}_{\mathsf{F}}\colon (\mathit{Val} \to \mathit{List\ Val} \to \mathit{iProp}) \to \mathit{Val} \to \mathit{List\ Val} \to \mathit{iProp}$. But we add an extra pointwise to the left and right-hand side of the respectful relation for the extra argument.

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

We can thus write down the monotonicity of a function without explicit currying and uncurrying.

Monotone proof search The monotone proof search is based on identifying the top-level relation and the top-level function beneath it. Thus, in the below proof state, the magic wand is the top-level relation and the disjunction is the top-level function.

$$\overset{\text{Top-level function}}{\square(\cdots) \vdash (\cdots \overset{\swarrow}{\vee} \cdots) \underset{\underset{\text{Top-level relation}}{\uparrow}}{\mathrel{-\!\!*}} (\cdots \overset{\searrow}{\vee} \cdots)}$$

Using these descriptions we show a proof using our monotone proof search. Then, we outline the steps we took in this proof.

Example 3.12: $\mathsf{isMLL_F}$ is monotone

The predicate $\mathsf{isMLL_F}$: $(Val \to List\ Val \to iProp) \to Val \to List\ Val \to iProp$ is monotone in its first argument. Thus, $\mathsf{isMLL_F}$ is a proper element of

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

In other words

$$\square\,(\forall hd\,\vec{v}.\,\Phi\,hd\,\vec{v} \mathrel{-\!\!*} \Psi\,hd\,\vec{v}) \mathrel{-\!\!*} \forall hd\,\vec{v}.\,\mathsf{isMLL_F}\,\Phi\,hd\,\vec{v} \mathrel{-\!\!*} \mathsf{isMLL_F}\,\Psi\,hd\,\vec{v}$$

Proof. We assume any premises, $\square\,(\forall hd\,\vec{v}.\,\Phi\,hd\,\vec{v} \mathrel{-\!\!*} \Psi\,hd\,\vec{v})$. We omit the premises in future-proof states, but it is always there since it is persistent. Next, we introduce the universal quantifiers. After unfolding $\mathsf{isMLL_F}$, we have to prove the following.

$$(\cdots \vee \cdots \Phi \cdots) \mathrel{-\!\!*} (\cdots \vee \cdots \Psi \cdots)$$

Thus, the top-level connective is the wand and the one below it is the disjunction. The signature $(\mathrel{-\!\!*}) \Longrightarrow (\mathrel{-\!\!*}) \Longrightarrow (\mathrel{-\!\!*})$ ends on a magic wand and has the disjunction as a proper element. We apply $((\mathrel{-\!\!*}) \Longrightarrow (\mathrel{-\!\!*}) \Longrightarrow (\mathrel{-\!\!*}))(\vee)(\vee)$, resulting in two statements to prove.

$$\begin{aligned}
&(hd = \texttt{\textbf{none}} * \vec{v} = []) \mathrel{-\!\!*} (hd = \texttt{\textbf{none}} * \vec{v} = [])\\
&(\cdots \Phi \cdots \vee \cdots \Phi \cdots) \mathrel{-\!\!*} (\cdots \Psi \cdots \vee \cdots \Psi \cdots)
\end{aligned}$$

The first statement follows directly from reflexivity of the magic wand. The second statement utilizes the same disjunction signature again. Thus, we just show the result of applying it.

$$\begin{aligned}
&(\exists \ell, v', tl.\ \cdots \Phi \cdots) \mathrel{-\!\!*} (\exists \ell, v', tl.\ \cdots \Psi \cdots)\\
&(\exists \ell, v', \vec{v}'', tl.\ \cdots \Phi \cdots) \mathrel{-\!\!*} (\exists \ell, v', \vec{v}'', tl.\ \cdots \Psi \cdots)
\end{aligned}$$

Both statements have as top-level relation $(\mathrel{-\!\!*})$ with below it $\exists$. We apply the signature of $\exists$ with as result.

$$\begin{aligned}
&\forall \ell.\,(\exists v', tl.\ \cdots \Phi \cdots) \mathrel{-\!\!*} (\exists v', tl.\ \cdots \Psi \cdots)\\
&\forall \ell.\,(\exists v', \vec{v}'', tl.\ \cdots \Phi \cdots) \mathrel{-\!\!*} (\exists v', \vec{v}'', tl.\ \cdots \Psi \cdots)
\end{aligned}$$

We introduce ℓ and repeat these steps until the existential quantification is no longer the top-level function.

$$\begin{aligned}
&(hd = \texttt{\textbf{some}}\,\ell * \ell fmapsto(v', \texttt{\textbf{true}}, tl) * \Phi\,tl\,\vec{v}) \mathrel{-\!\!*}\\
&(hd = \texttt{\textbf{some}}\,\ell * \ell fmapsto(v', \texttt{\textbf{true}}, tl) * \Psi\,tl\,\vec{v})\\
&\begin{pmatrix} hd = \texttt{\textbf{some}}\,\ell * \ell fmapsto(v', \texttt{\textbf{false}}, tl) * \\ \vec{v} = v' :: \vec{v}'' * \Phi\,tl\,\vec{v}'' \end{pmatrix} \mathrel{-\!\!*}\\
&\begin{pmatrix} hd = \texttt{\textbf{some}}\,\ell * \ell fmapsto(v', \texttt{\textbf{false}}, tl) * \\ \vec{v} = v' :: \vec{v}'' * \Psi\,tl\,\vec{v}'' \end{pmatrix}
\end{aligned}$$

We can now repeatedly apply the signature of $(*)$ and apply reflexivity to any created propositions without Φ or Ψ. This leaves us with

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

These hold from the assumption. □

The strategy we use for proof search consists of two steps. We have a normalization step, and we have an application step.

Normalization Introduce any universal quantifiers, extra created wands and modalities. Afterward, do an application step.

Application We apply the first option that works.

1. If the left and right-hand side of the relation are equal, and the relation is reflexive, apply reflexivity.
2. Check if the conclusion follows from a premise, and then apply it.
3. Look for a signature of the top-level function where the last relation matches the top-level relation of the conclusion. Apply it if we find one. Next, do a normalization step.

We start the proof with the normalization step and continue until all created branches are proven.

Generating the fixpoints theorem Given the above proof of monotonicity of $\mathsf{isMLL_F}$, theorem 3.3 does not give the least fixpoint for $\mathsf{isMLL_F}$. We change the definition to add an arbitrary number of arguments to the fixpoint and its properties.

$$\mu\mathsf{F}\,x_1\,\cdots\,x_n \triangleq \forall\Phi.\;\Box(\forall y_1,\cdots,y_n.\,\mathsf{F}\,\Phi\,y_1\,\cdots\,y_n \mathrel{-\!\!*} \Phi\,y_1\,\cdots\,y_n) \mathrel{-\!\!*} \Phi\,x_1\,\cdots\,x_n$$

The above definition cannot be written in Coq, since a definition has to have a concrete arity. Thus, we generate the least fixpoint theorem for any function we want to take the least fixpoint of.

Example 3.13: isMLL least fixpoint theorem

We have the monotone function

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

We use the above definition of the least fixpoint with $n = 2$.

$$\mu\mathsf{isMLL_F}\,hd\,\vec{v} \triangleq \forall\Phi.\;\Box(\forall hd', \vec{v}'.\,\mathsf{isMLL_F}\,\Phi\,hd'\,\vec{v}' \mathrel{-\!\!*} \Phi\,hd'\,\vec{v}') \mathrel{-\!\!*} \Phi\,hd\,\vec{v}$$

For the induction principle, we apply the same strategy. With isMLL, we get the induction principle as described in example 3.5.

The generation of these different theorems and definitions will be done using Elpi as is explained in the next two chapters.