# Effectful realizability

## MFoCS Seminar

Ties Steijn

Radboud University

January 2026

Papers by Cohen, Grunfeld, Kirst, Miquey, Tate

# Overview

- Realizability: connection between proofs and programs.
- Example: program extraction (in Rocq).
- Target language is usually pure (no effects).
- Effects allow us to find realizers for more statements.
- Syntactic approach: **EffHOL**.
  - Paper: *Syntactic Effectful Realizability in Higher-Order Logic*. L. Cohen, A. Grunfeld, D. Kirst, E. Miquey, 2025.
- Semantic approach: **Evidenced frames**.
  - Paper: *Evidenced Frames: A Unifying Framework Broadening Realizability Models*. L. Cohen, E. Miquey, R. Tate, 2021.
- Connection to existing theory: **Tripos** (and topos) theory.

# Overview

- **HOL** is like predicate logic with the addition of powerset types.
- This lets us describe predicates about propositions, predicates about predicates, . . .

# Overview

- **HOL** is like predicate logic with the addition of powerset types.
- This lets us describe predicates about propositions, predicates about predicates, . . .
- **EffHOL** is a system comprised of (higher-order) logic rules and an effectful programming language.

# Overview

- **HOL** is like predicate logic with the addition of powerset types.
- This lets us describe predicates about propositions, predicates about predicates, . . .
- **EffHOL** is a system comprised of (higher-order) logic rules and an effectful programming language.
- **Evidenced frames** are defined by an *evidence relation* $\phi_1 \xrightarrow{e} \phi_2$.
  - Essentially a proof relevant ordering on propositions.
  - The evidence can be taken as effectful programs.

# Overview

- **HOL** is like predicate logic with the addition of powerset types.
- This lets us describe predicates about propositions, predicates about predicates, . . .
- **EffHOL** is a system comprised of (higher-order) logic rules and an effectful programming language.
- **Evidenced frames** are defined by an *evidence relation* $\phi_1 \xrightarrow{e} \phi_2$.
  - Essentially a proof relevant ordering on propositions.
  - The evidence can be taken as effectful programs.
- **Triposes** are category-theoretical models of **HOL**.
- **Toposes** are more elaborate category-theoretical models of **HOL**.
  - The long-time standard.
  - Beyond the scope of this presentation.

# Overview

- **HOL** is like predicate logic with the addition of powerset types.
- This lets us describe predicates about propositions, predicates about predicates, . . .
- **EffHOL** is a system comprised of (higher-order) logic rules and an effectful programming language.
- **Evidenced frames** are defined by an *evidence relation* $\phi_1 \xrightarrow{e} \phi_2$.
  - Essentially a proof relevant ordering on propositions.
  - The evidence can be taken as effectful programs.
- **Triposes** are category-theoretical models of **HOL**.
- **Toposes** are more elaborate category-theoretical models of **HOL**.
  - The long-time standard.
  - Beyond the scope of this presentation.

We'll show that both **EffHOL** instances and evidenced frames can be translated to triposes (and hence toposes).

$$\textbf{HOL} \xrightarrow[\text{Syntactic translation}]{} \textbf{EffHOL} \xrightarrow[\text{Semantics}]{} \textbf{Evidenced frame} \xrightarrow[\text{Uniform families}]{} \textbf{Tripos} \xrightarrow[\text{tripos-to-topos}]{} \textbf{Topos}$$

# Higher-Order Logic

- Sorts: $s = \star \mid s \to \star$.
  - Think of $\texttt{Prop}$, $\mathcal{P}(\texttt{Prop})$, $\mathcal{P}(\mathcal{P}(\texttt{Prop}))$, $\ldots$.

# Higher-Order Logic

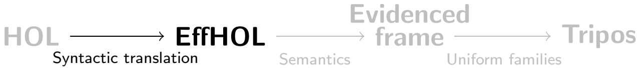
- Sorts: $s = \star \mid s \to \star$.
  - Think of Prop, $\mathcal{P}(\texttt{Prop})$, $\mathcal{P}(\mathcal{P}(\texttt{Prop}))$, ....
- Formulas: $\varphi = \forall x : s, \varphi \mid \varphi \sqsupset \varphi \mid t \in t \mid \bar{t}$.
- Terms: $t = x \mid \{x : s \mid \varphi\} \mid \{x\}$.

# Higher-Order Logic

- Sorts: $s = \star \mid s \to \star$.
  - Think of $\texttt{Prop}$, $\mathcal{P}(\texttt{Prop})$, $\mathcal{P}(\mathcal{P}(\texttt{Prop}))$, $\ldots$.
- Formulas: $\varphi = \forall x : s, \varphi \mid \varphi \sqsupset \varphi \mid t \in t \mid \bar{t}$.
- Terms: $t = x \mid \{x : s \mid \varphi\} \mid \{x\}$.
- Deduction rules:
  - Usual logic rules (first order).
  - Rules for comprehension terms:

$$\overline{\{\varphi\}} \iff \varphi$$
$$t \in \{x : s \mid \varphi\} \iff \varphi[x := t]$$

# Overview

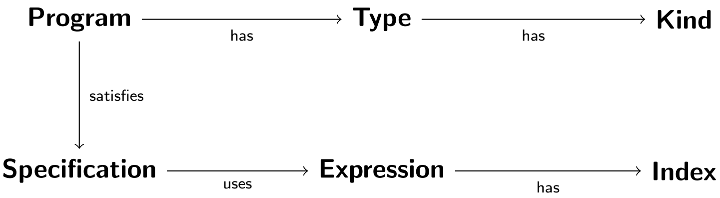

HOL
Syntactic translation
EffHOL
Semantics
Evidenced frame
Uniform families
Tripos

# EffHOL: overview

- We would like to translate HOL proofs to programs in some language.
- This language will be an effectful version of $\lambda\omega$.
- **Idea:** for any HOL proposition, the set of programs of the corresponding type are the *potential realizers*.
- The programs that additionally satisfy the corresponding specification are *actual realizers*.
- Note the similarity to Kreisel's *modified realizability*.
- **EffHOL** combines an (effectful) programming language with a logic system.

Components of **EffHOL**:

- *Kinds*: correspond to HOL sorts.
- *Types* and *Specifications*: correspond to HOL propositions.
- *Programs*: correspond to HOL proofs.
- *Expressions*: correspond to HOL terms.
- *Indices*: the types of expressions.



# EffHOL: overview

- Remember: Haskell implements effects via the `Monad` typeclass.
- In **EffHOL** we use a similar idea.
- Key difference: there is a primitive computation type $M(\tau)$.
- Programs may use the monadic constructs `return` and `bind`.
- How do we reason about values returned by effectful computations?

# EffHOL: overview

- Remember: Haskell implements effects via the `Monad` typeclass.
- In **EffHOL** we use a similar idea.
- Key difference: there is a primitive computation type $M(\tau)$.
- Programs may use the monadic constructs `return` and `bind`.
- How do we reason about values returned by effectful computations?
- They may not be deterministic, or they may fail altogether.
- Solution: specifications may use **modality** to handle the results of monadic computations.
- If $p$ evaluates to $x$, then $\varphi(x)$ holds.

# EffHOL: syntax

- **Kinds:** same as in **HOL**.

$$\kappa = \star \mid \kappa \to \star$$

# EffHOL: syntax

- **Kinds:** same as in **HOL**.

$$\kappa = \star \mid \kappa \to \star$$

- **Types:** as in $\lambda\omega$, with an additional computation type constructor $M(\tau)$.

$$\tau = X \mid \tau \to \tau$$

# EffHOL: syntax

- **Kinds:** same as in **HOL**.

$$\kappa = \star \mid \kappa \to \star$$

- **Types:** as in $\lambda\omega$, with an additional computation type constructor $M(\tau)$.

$$\tau = X \mid \tau \to \tau \mid \prod X : \kappa.\tau \mid \tau\ \tau$$

# EffHOL: syntax

- **Kinds:** same as in **HOL**.

$$\kappa = \star \mid \kappa \to \star$$

- **Types:** as in $\lambda\omega$, with an additional computation type constructor $M(\tau)$.

$$\tau = X \mid \tau \to \tau \mid \prod X : \kappa.\tau \mid \tau\ \tau \mid \overline{\Lambda} X : \kappa.\tau$$

# EffHOL: syntax

- **Kinds:** same as in **HOL**.

$$\kappa = \star \mid \kappa \to \star$$

- **Types:** as in $\lambda\omega$, with an additional computation type constructor $M(\tau)$.

$$\tau = X \mid \tau \to \tau \mid \prod X : \kappa.\tau \mid \tau\ \tau \mid \overline{\Lambda} X : \kappa.\tau \mid M(\tau)$$

# EffHOL: syntax

- **Kinds:** same as in **HOL**.

$$\kappa = \star \mid \kappa \to \star$$

- **Types:** as in $\lambda\omega$, with an additional computation type constructor $M(\tau)$.

$$\tau = X \mid \tau \to \tau \mid \prod X : \kappa.\tau \mid \tau\ \tau \mid \overline{\Lambda} X : \kappa.\tau \mid M(\tau)$$

- **Programs:** as in $\lambda\omega$, with `return` and `bind`.

$$p = x \mid p\ p \mid \lambda x : \tau.p$$

# EffHOL: syntax

- **Kinds:** same as in **HOL**.

$$\kappa = \star \mid \kappa \to \star$$

- **Types:** as in $\lambda\omega$, with an additional computation type constructor $M(\tau)$.

$$\tau = X \mid \tau \to \tau \mid \prod X : \kappa.\tau \mid \tau\ \tau \mid \overline{\Lambda} X : \kappa.\tau \mid M(\tau)$$

- **Programs:** as in $\lambda\omega$, with `return` and `bind`.

$$p = x \mid p\ p \mid \lambda x : \tau.p \mid p\ \tau \mid \Lambda X : \kappa.p$$

# EffHOL: syntax

- **Kinds:** same as in **HOL**.

$$\kappa = \star \mid \kappa \to \star$$

- **Types:** as in $\lambda\omega$, with an additional computation type constructor $M(\tau)$.

$$\tau = X \mid \tau \to \tau \mid \prod X : \kappa.\tau \mid \tau\ \tau \mid \overline{\Lambda} X : \kappa.\tau \mid M(\tau)$$

- **Programs:** as in $\lambda\omega$, with `return` and `bind`.

$$p = x \mid p\ p \mid \lambda x : \tau.p \mid p\ \tau \mid \Lambda X : \kappa.p \mid [p] \mid \text{let } x \leftarrow p \text{ in } p$$

# EffHOL: syntax

**Specifications:**

- Implication: $\varphi \supset \varphi$.
- Universal quantification over programs/types/expressions:
  $\sqcap x : \tau.\varphi \mid \cap X : \kappa.\varphi \mid \forall y : \sigma.\varphi$.

# EffHOL: syntax

**Specifications:**

- Implication: $\varphi \supset \varphi$.
- Universal quantification over programs/types/expressions: $\sqcap x : \tau.\varphi \mid \cap X : \kappa.\varphi \mid \forall y : \sigma.\varphi$.
- Expressions are like comprehensions in **HOL**.
- Difference: they additionally depend on a program of a certain type.
- Membership: $p \Subset e \mid p; e \Subset e$.

# EffHOL: syntax

**Specifications:**

- Implication: $\varphi \supset \varphi$.
- Universal quantification over programs/types/expressions: $\sqcap x : \tau.\varphi \mid \cap X : \kappa.\varphi \mid \forall y : \sigma.\varphi$.
- Expressions are like comprehensions in **HOL**.
- Difference: they additionally depend on a program of a certain type.
- Membership: $p \mathrel{\in\kern-0.6em\shortmid} e \mid p; e \mathrel{\in\kern-0.6em\shortmid} e$.
- Modality: $\langle x \leftarrow p \rangle\ \varphi$.
  - Intuition: if $p$ evaluates to $x$, then $\varphi(x)$ holds.

# EffHOL: syntax

- **Expressions:** like the comprehensions in **HOL**.
- Since specifications are about programs, they also depend on a program of a given type.

$$e = y \mid \{x : \tau \mid \varphi\} \mid \{x : \tau, y : \sigma \mid \varphi\}$$

# EffHOL: syntax

- **Expressions:** like the comprehensions in **HOL**.
- Since specifications are about programs, they also depend on a program of a given type.

$$e = y \mid \{x : \tau \mid \varphi\} \mid \{x : \tau, y : \sigma \mid \varphi\} \mid \Lambda X : \kappa . e \mid e\ \tau$$

Polymorphic expressions are needed to handle higher-order comprehensions in **HOL**.

# EffHOL: syntax

- **Expressions:** like the comprehensions in **HOL**.
- Since specifications are about programs, they also depend on a program of a given type.

$$e = y \mid \{x : \tau \mid \varphi\} \mid \{x : \tau, y : \sigma \mid \varphi\} \mid \wedge X : \kappa.e \mid e\ \tau$$

  Polymorphic expressions are needed to handle higher-order comprehensions in **HOL**.
- **Indices:**
  - Intuition: read $R_\tau$ as $\mathcal{P}(\tau)$.
  - $R_\tau$: type of $\{x : \tau \mid \varphi\}$.
  - $R_\tau(\sigma)$: type of $\{x : \tau, y : \sigma \mid \varphi\}$.
  - $\bigwedge X : \kappa.\ \sigma$: type of $\wedge X : \kappa.\ e$.

# EffHOL: deduction rules

- Rules for assumption, introduction/elimination of $\supset$ and all three quantifiers.
- Introduction/elimination rules for comprehensions.
- Conversion rules for programs/types.
- Rules to handle monadic computation:

# EffHOL: deduction rules

- Rules for assumption, introduction/elimination of $\supset$ and all three quantifiers.
- Introduction/elimination rules for comprehensions.
- Conversion rules for programs/types.
- Rules to handle monadic computation:
  - Rule for `return`:
    $\varphi[x := p] \implies \langle x \leftarrow [p] \rangle \; \varphi$.

# EffHOL: deduction rules

- Rules for assumption, introduction/elimination of $\supset$ and all three quantifiers.
- Introduction/elimination rules for comprehensions.
- Conversion rules for programs/types.
- Rules to handle monadic computation:
  - Rule for `return`:
    $\varphi[x := p] \implies \langle x \leftarrow [p] \rangle\ \varphi$.
  - Rule for `bind`:
    $\langle x_1 \leftarrow p_1 \rangle\ \langle x_2 \leftarrow p_2 \rangle\ \varphi \implies \langle x_2 \leftarrow \text{let } x_1 \leftarrow p_1 \text{ in } p_2 \rangle\ \varphi$.

# EffHOL: deduction rules

- Rules for assumption, introduction/elimination of $\supset$ and all three quantifiers.
- Introduction/elimination rules for comprehensions.
- Conversion rules for programs/types.
- Rules to handle monadic computation:
  - Rule for `return`:
    $\varphi[x := p] \implies \langle x \leftarrow [p] \rangle\ \varphi$.
  - Rule for `bind`:
    $\langle x_1 \leftarrow p_1 \rangle\ \langle x_2 \leftarrow p_2 \rangle\ \varphi \implies \langle x_2 \leftarrow \text{let } x_1 \leftarrow p_1 \text{ in } p_2 \rangle\ \varphi$.
  - $\supset$-elim inside modality:
    $\varphi_1 \supset \varphi_2, \langle x \leftarrow p \rangle\ \varphi_1 \implies \langle x \leftarrow p \rangle\ \varphi_2$.

# Translating HOL to EffHOL

Components of the translation:

- $[\![\_]\!]^K : \texttt{sort} \to \texttt{kind}$
- $[\![\_]\!]^I_\_ : \texttt{sort} \to \texttt{type} \to \texttt{index}$
- $[\![\_]\!]^T : \texttt{prop} \to \texttt{type}$
  - $[\![\_]\!]^t : \texttt{term} \to \texttt{type}$
- $[\![\_]\!]^S_\_ : \texttt{prop} \to \texttt{prog} \to \texttt{spec}$
  - $[\![\_]\!]^e : \texttt{term} \to \texttt{expr}$

We'll only look at the type and specification translations.

# Translating HOL to EffHOL

Translation of implication:

$$\llbracket \psi_1 \sqsupset \psi_2 \rrbracket^T = \llbracket \psi_1 \rrbracket^T \to M(\llbracket \psi_2 \rrbracket^T)$$
$$\llbracket \psi_1 \sqsupset \psi_2 \rrbracket^S_p = \sqcap x_1 : \llbracket \psi_1 \rrbracket^T .\ \llbracket \psi_1 \rrbracket^S_{x_1} \supset \langle x_2 \leftarrow p\ x_1 \rangle\ \llbracket \psi_2 \rrbracket^S_{x_2}$$

# Translating HOL to EffHOL

Translation of implication:

$$\llbracket \psi_1 \supset \psi_2 \rrbracket^T = \llbracket \psi_1 \rrbracket^T \to M(\llbracket \psi_2 \rrbracket^T)$$
$$\llbracket \psi_1 \supset \psi_2 \rrbracket^S_p = \sqcap x_1 : \llbracket \psi_1 \rrbracket^T.\ \llbracket \psi_1 \rrbracket^S_{x_1} \supset \langle x_2 \leftarrow p\ x_1 \rangle\ \llbracket \psi_2 \rrbracket^S_{x_2}$$

Translation of universal quantification:

$$\llbracket \forall x : s.\psi \rrbracket^T = \prod X_x : \llbracket s \rrbracket^K.\ M(\llbracket \psi \rrbracket^T)$$
$$\llbracket \forall x : s.\psi \rrbracket^S_p = \cap X_x : \llbracket s \rrbracket^K.\ \forall y_x : \llbracket s \rrbracket^I_{X_x}.\ \langle x_0 \leftarrow p\ X_x \rangle\ \llbracket \psi \rrbracket^S_{x_0}$$

The quantification over $y_x$ is needed to handle occurrences of $x$ in $\psi$: a base case translates these to $y_x$.

# Translating HOL to EffHOL: example

Consider the following **HOL** proposition:

$$\forall a : \star.\ \overline{a} \sqsupset \overline{a}$$

The corresponding **EffHOL** type is:

$$\prod A : \star.\ M(A \to M(A))$$

# Translating HOL to EffHOL: example

Consider the following **HOL** proposition:

$$\forall a : \star.\ \overline{a} \sqsupset \overline{a}$$

The corresponding **EffHOL** type is:

$$\prod A : \star.\ M(A \to M(A))$$

Programs $p$ of this type should satisfy this **EffHOL** specification:

$$\cap A : \star.\ \forall S : R_A.\ \langle f \leftarrow p\ A \rangle \sqcap_{x:A}.\ x \Subset S \supset \langle y \leftarrow f\ x \rangle\ y \Subset S.$$

**(Approximate) meaning:** $p$ is a polymorphic computation that transforms programs $x : A$ that are in $S$ into programs $y : A$ also in $S$.

# Translating HOL to EffHOL: proofs

**Theorem (Soundness)**

For any **HOL** proof of a theorem $\psi$, we can construct a program $p$ such that

- $p : M[\![\psi]\!]^T$,
- $\langle x_r \leftarrow p \rangle \; [\![\psi]\!]^S_{x_r}$.

**Proof:** A rule-by-rule construction of $p$ from the proof of $\psi$.

# Instances of EffHOL

- Define **EffHOL**$^-$ as the fragment of **EffHOL** with all monad-related constructs removed.
- A *pure instance* of **EffHOL** is an interpretation of **EffHOL** in **EffHOL**$^-$ that
  - gives an interpretation of the monadic constructs that does not use $M(\tau)$, `return`, `bind`,
  - possibly extends the reduction relation on programs.

# Example: memoization and Countable Choice

- Consider the axiom of Countable Choice (CC):
  *Any total relation* $u \subseteq \mathbb{N} \times \tau$ *has a deterministic total subrelation.*
- CC is true if computations are deterministic.
- CC may be false if computations are nondeterministic.

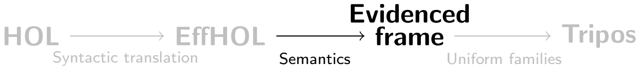
# Example: memoization and Countable Choice

- Consider the axiom of Countable Choice (CC):
  *Any total relation $u \subseteq \mathbb{N} \times \tau$ has a deterministic total subrelation.*
- CC is true if computations are deterministic.
- CC may be false if computations are nondeterministic.
- Now suppose computations are nondeterministic, but we keep track of a program $p : \tau$ for every natural number.
- $M(\tau)$ includes a state of the form $\mathbb{N} \to \tau$.
- Let $\texttt{lookup}_n\ p$ be a program that looks up the program stored at $n$ and
  - Returns it if it exists.
  - Returns and sets $p$ if it does not exist.

# Example: memoization and Countable Choice

- Consider the axiom of Countable Choice (CC):
  *Any total relation $u \subseteq \mathbb{N} \times \tau$ has a deterministic total subrelation.*
- CC is true if computations are deterministic.
- CC may be false if computations are nondeterministic.
- Now suppose computations are nondeterministic, but we keep track of a program $p : \tau$ for every natural number.
- $M(\tau)$ includes a state of the form $\mathbb{N} \to \tau$.
- Let $\texttt{lookup}_n\ p$ be a program that looks up the program stored at $n$ and
  - Returns it if it exists.
  - Returns and sets $p$ if it does not exist.
- Use the state to keep track of the first program we find that realizes $(n, x) \in u$ (using `lookup`) and always return that program.
- We can now realize CC, even if computations are nondeterministic.

# Overview


HOL
Syntactic translation
EffHOL
Semantics
Evidenced frame
Uniform families
Tripos

# Evidenced frame: overview

- **Idea:** View entailment as an ordering.
- The ordering is proof relevant: we have an *evidence relation* $\phi_1 \xrightarrow{e} \phi_2$.
- What should we take as evidence?

# Evidenced frame: overview

- **Idea:** View entailment as an ordering.
- The ordering is proof relevant: we have an *evidence relation* $\phi_1 \xrightarrow{e} \phi_2$.
- What should we take as evidence?
- A natural choice would be elements of a PCA, but we would like effects.
- *Computational systems* are an effectful version of PCAs.

# Evidenced frame: definition

An evidenced frame is a triple $(\Phi, E, \cdot \overset{\cdot}{\rightarrow} \cdot)$ with the following properties:

- **Reflexivity:** Evidence $e_{\mathsf{id}}$ such that $\phi \overset{e_{\mathsf{id}}}{\rightarrow} \phi$.
- **Transitivity:** An operator $; \in E \rightarrow E \rightarrow E$ such that:
  If $\phi_1 \overset{e}{\rightarrow} \phi_2$ and $\phi_2 \overset{e'}{\rightarrow} \phi_3$, then $\phi_1 \overset{e;e'}{\rightarrow} \phi_3$.
- **Top:** A proposition $\top$ and evidence $e_\top$ such that $\phi \overset{e_\top}{\rightarrow} \top$.

# Evidenced frame: definition

An evidenced frame is a triple $(\Phi, E, \cdot \overset{\cdot}{\rightarrow} \cdot)$ with the following properties:

- **Reflexivity:** Evidence $e_{\mathsf{id}}$ such that $\phi \overset{e_{\mathsf{id}}}{\rightarrow} \phi$.
- **Transitivity:** An operator $; \in E \rightarrow E \rightarrow E$ such that:
  If $\phi_1 \overset{e}{\rightarrow} \phi_2$ and $\phi_2 \overset{e'}{\rightarrow} \phi_3$, then $\phi_1 \overset{e;e'}{\rightarrow} \phi_3$.
- **Top:** A proposition $\top$ and evidence $e_\top$ such that $\phi \overset{e_\top}{\rightarrow} \top$.
- **Conjunction:** Operators $\langle \cdot, \cdot \rangle \in E \rightarrow E \rightarrow E$, $\wedge \in \Phi \rightarrow \Phi \rightarrow \Phi$ and evidence $e_{\mathsf{fst}}, e_{\mathsf{snd}}$ such that:

$$\phi_1 \wedge \phi_2 \overset{e_{\mathsf{fst}}}{\rightarrow} \phi_1, \qquad \phi_1 \wedge \phi_2 \overset{e_{\mathsf{snd}}}{\rightarrow} \phi_2$$

If $\phi \overset{e_1}{\rightarrow} \phi_1$ and $\phi \overset{e_2}{\rightarrow} \phi_2$, then $\phi \overset{\langle e_1, e_2 \rangle}{\rightarrow} \phi_1 \wedge \phi_2$

# Evidenced frame: definition

- **Universal implication:** A combination of implication and universal quantification.
  There are operators $\supset \in \Phi \times \mathcal{P}(\Phi) \to \Phi, \lambda \in E \to E$ and evidence $e_{\text{eval}}$ such that:
  - If for all $\phi \in \overrightarrow{\phi}$ we have $\phi_1 \wedge \phi_2 \xrightarrow{e} \phi$, then $\phi_1 \xrightarrow{\lambda e} \phi_2 \supset \overrightarrow{\phi}$.
  - For all $\phi \in \overrightarrow{\phi}$ we have $(\phi_1 \supset \overrightarrow{\phi}) \wedge \phi_1 \xrightarrow{e_{\text{eval}}} \phi$.

  How do we get regular implication and universal quantification?

# Evidenced frame: definition

- **Universal implication:** A combination of implication and universal quantification.
  There are operators $\supset \in \Phi \times \mathcal{P}(\Phi) \to \Phi, \lambda \in E \to E$ and evidence $e_{\mathsf{eval}}$ such that:
  - If for all $\phi \in \overrightarrow{\phi}$ we have $\phi_1 \wedge \phi_2 \xrightarrow{e} \phi$, then $\phi_1 \xrightarrow{\lambda e} \phi_2 \supset \overrightarrow{\phi}$.
  - For all $\phi \in \overrightarrow{\phi}$ we have $(\phi_1 \supset \overrightarrow{\phi}) \wedge \phi_1 \xrightarrow{e_{\mathsf{eval}}} \phi$.

  How do we get regular implication and universal quantification?
- Implication: $\phi_1 \supset \{\phi_2\}$.
- Universal quantification: $\top \supset \overrightarrow{\phi}$.
  - The *variable condition* is hidden: holds because the same evidence must work for all $\phi \in \overrightarrow{\phi}$.

# Computational systems

- The definition suggests that we would like codes behaving like $\lambda$ terms.
- As an example we will consider *computational systems*: an effectful generalization of PCAs.

# Computational systems

- The definition suggests that we would like codes behaving like $\lambda$ terms.
- As an example we will consider *computational systems*: an effectful generalization of PCAs.
- Partial applicative structure: set with a partial application operation, i.e. either $c \cdot c' \uparrow$ or $c \cdot c' \downarrow d$.
- Partial combinatory algebra: PAS that has elements behaving like $\lambda$-terms.
- Example: $\mathbb{N}$ with Kleene application: $n \cdot m$ is the $n$-th Turing machine applied to $m$.

# Computational systems

- Instead of one partial reduction relation $\downarrow$, have **stateful** termination and reduction relations:
  - $\sigma \leq \sigma'$: $\sigma'$ is a possible future of $\sigma$.
  - $c \downarrow^{\sigma}$: $c$ terminates in state $\sigma$.
  - $c \downarrow^{\sigma}_{\sigma'} c'$: $c$ reduces to $c'$ in state $\sigma$, changing it to $\sigma'$.
- A code may reduce to multiple different codes from the same state, or no codes at all: we may have nondeterminism, failure.

# Computational systems

- Instead of one partial reduction relation $\downarrow$, have **stateful** termination and reduction relations:
  - $\sigma \leq \sigma'$: $\sigma'$ is a possible future of $\sigma$.
  - $c \downarrow^{\sigma}$: $c$ terminates in state $\sigma$.
  - $c \downarrow^{\sigma}_{\sigma'} c'$: $c$ reduces to $c'$ in state $\sigma$, changing it to $\sigma'$.
- A code may reduce to multiple different codes from the same state, or no codes at all: we may have nondeterminism, failure.
- Example:

$$\begin{aligned}
&\texttt{lookup}_n \cdot c \downarrow^{\sigma} && \\
&\texttt{lookup}_n \cdot c \downarrow^{\sigma}_{\sigma} c' && \text{if } n \mapsto c' \in \sigma \\
&\texttt{lookup}_n \cdot c \downarrow^{\sigma}_{\sigma, n \mapsto c} c && \text{if there is no } c' \text{ such that } n \mapsto c' \in \sigma
\end{aligned}$$

$\texttt{lookup}_n \cdot c$ looks up the code stored at $n$ and returns it, or if that fails, returns and sets $c$.

# Evidenced frame for a computational system

- Let $(C, \Sigma)$ be a computational system.
- Propositions are stateful predicates, $\phi \subseteq \Sigma \times C$ that are **future-stable**.
- Suppose for now that $C$ is the set of evidence.
  - In reality, we have to be careful with the codes we include: some may make the evidenced frame inconsistent.
- We say that $\phi_1 \xrightarrow{e} \phi_2$ if the following holds:
  If $(\sigma, c) \in \phi_1$, then:
  - $e \cdot c \downarrow^{\sigma}$.
  - If $e \cdot c \downarrow^{\sigma}_{\sigma'} c'$, then $(\sigma', c') \in \phi_2$.

# Evidenced frame for a computational system

- Let $(C, \Sigma)$ be a computational system.
- Propositions are stateful predicates, $\phi \subseteq \Sigma \times C$ that are **future-stable**.
- Suppose for now that $C$ is the set of evidence.
  - In reality, we have to be careful with the codes we include: some may make the evidenced frame inconsistent.
- We say that $\phi_1 \xrightarrow{e} \phi_2$ if the following holds:
  If $(\sigma, c) \in \phi_1$, then:
  - $e \cdot c \downarrow^{\sigma}$.
  - If $e \cdot c \downarrow^{\sigma}_{\sigma'} c'$, then $(\sigma', c') \in \phi_2$.
- $\phi_1 \wedge \phi_2$ consists of pairs of codes.
- $\phi_1 \supset \overrightarrow{\phi}$ consists of codes that work as evidence $\phi_1 \xrightarrow{c} \phi$ for all $\phi \in \overrightarrow{\phi}$.

# Evidenced frame for EffHOL instances

- Consider a pure instance of **EffHOL**.
- Idea: A **HOL** proposition $\varphi$ is realized by the set of closed programs $p : [\![\varphi]\!]^T$ such that $[\![\varphi]\!]^S_p$ holds.
- The evidence relation will resemble $\forall p : [\![\varphi]\!]^T.\ [\![\varphi]\!]^S_p \to \langle x \leftarrow e\ p \rangle\ [\![\psi]\!]^S_x$.

# Evidenced frame for EffHOL instances

- Consider a pure instance of **EffHOL**.
- Idea: A **HOL** proposition $\varphi$ is realized by the set of closed programs $p : [\![\varphi]\!]^T$ such that $[\![\varphi]\!]^S_p$ holds.
- The evidence relation will resemble $\forall p : [\![\varphi]\!]^T.\ [\![\varphi]\!]^S_p \rightarrow \langle x \leftarrow e\ p \rangle\ [\![\psi]\!]^S_x$.
- We cannot get a tripos (and hence an evidenced frame) unless we erase all types [Lietz, Streicher, 2002].
- Let $\lfloor p \rfloor$ be the program $p$ with all type annotations, type abstractions and type applications removed.
- Define $\mathcal{P}$ as the set of all (closed) programs in **EffHOL**.
- Define $\Lambda = \{\lfloor p \rfloor \mid p \in \mathcal{P}\}$.
- Define $\mathcal{V}$ as the set of *values* in $\Lambda$.

# Evidenced frame for EffHOL instances

- Define $\Phi_{\mathsf{ef}} = \{\lfloor P \rfloor \mid P \subseteq \mathcal{P},\ \lfloor P \rfloor \subseteq \mathcal{V}\}$.
- Define $E_{\mathsf{ef}} = \Lambda$.
- For $\phi_1 \xrightarrow{e} \phi_2$, we would like to write $\forall p \in \phi_1.\ \langle x \leftarrow e\ p \rangle\ x \in \phi_2$.
- We need to **lift** sets of values to sets of programs that evaluate to those values.

# Evidenced frame for EffHOL instances

- Define $\Phi_{\text{ef}} = \{\lfloor P \rfloor \mid P \subseteq \mathcal{P},\ \lfloor P \rfloor \subseteq \mathcal{V}\}$.
- Define $E_{\text{ef}} = \Lambda$.
- For $\phi_1 \xrightarrow{e} \phi_2$, we would like to write $\forall p \in \phi_1.\ \langle x \leftarrow e\ p \rangle\ x \in \phi_2$.
- We need to **lift** sets of values to sets of programs that evaluate to those values.
- Pure instance, so the modality is just a (pure) specification.
- We can perform the lifting by replacing all the logical constructs to their meta counterparts.

**Theorem**

$(\Phi_{ef}, E_{ef}, \cdot \dot{\rightarrow} \cdot)$ *is an evidenced frame.*

# Overview

**HOL** $\longrightarrow$ (Syntactic translation) **EffHOL** $\longrightarrow$ (Semantics) **Evidenced frame** $\longrightarrow$ (Uniform families) **Tripos**

# Tripos: overview

- Model of **HOL** based on category theory.
- Highly abstract.
- The propositional part of the logic is similar to an evidenced frame: entailment is an ordering.
- **Difference:** the ordering is proof irrelevant.

# Tripos: overview

- Model of **HOL** based on category theory.
- Highly abstract.
- The propositional part of the logic is similar to an evidenced frame: entailment is an ordering.
- **Difference:** the ordering is proof irrelevant.
- This construction is known as a **Heyting algebra**.
  - Ordering $\leq$ represents entailment.
  - Operations $\wedge, \vee, \Rightarrow$, elements $\top, \bot$.

# Tripos: overview

- Model of **HOL** based on category theory.
- Highly abstract.
- The propositional part of the logic is similar to an evidenced frame: entailment is an ordering.
- **Difference:** the ordering is proof irrelevant.
- This construction is known as a **Heyting algebra**.
  - Ordering $\leq$ represents entailment.
  - Operations $\wedge, \vee, \Rightarrow$, elements $\top, \bot$.
- Triposes are generally defined using Heyting *pre*algebras: $\leq$ is not antisymmetric.
- Other components:
  - Predicate logic and quantifiers.
  - Higher-order logic.

# Tripos: definition

- Propositional logic is implemented through Heyting prealgebras.
- Predicate logic is implemented through a functor $\mathcal{T} : \mathbf{Set}^{\mathsf{op}} \to \mathbf{pHA}$.
  - $\mathcal{T}(\Gamma)$: predicates in context $\Gamma$.
  - Types interpreted as sets, terms are functions $\Gamma \to A$.
  - Functions $\Gamma \to \Gamma'$ induce a substitution on predicates $s^* : \mathcal{T}(\Gamma') \to \mathcal{T}(\Gamma)$.
  - Quantifiers are defined using a category-theoretical trick: **adjoints**.

# Tripos: definition

- Propositional logic is implemented through Heyting prealgebras.
- Predicate logic is implemented through a functor $\mathcal{T} : \mathbf{Set}^{\mathsf{op}} \to \mathbf{pHA}$.
  - $\mathcal{T}(\Gamma)$: predicates in context $\Gamma$.
  - Types interpreted as sets, terms are functions $\Gamma \to A$.
  - Functions $\Gamma \to \Gamma'$ induce a substitution on predicates $s^* : \mathcal{T}(\Gamma') \to \mathcal{T}(\Gamma)$.
  - Quantifiers are defined using a category-theoretical trick: **adjoints**.
- Higher order logic is implemented through a *generic predicate*.
  - $\Omega \in \mathbf{Set}$: functions like `Prop`.
  - Power set type: $A \to \Omega$.
  - $\chi_\phi : A \to \Omega$: corresponds to the comprehension $\{x : A \mid \phi\}$.
  - holds $\in \mathcal{T}(\Omega)$: implements membership, holds$(p(x))$ corresponds to $x \in p$.

# Tripos: quantifiers as adjoints

For any $s : \Gamma \to \Gamma'$, there is a **pHA**-morphism $\Pi_s : \mathcal{T}(\Gamma) \to \mathcal{T}(\Gamma')$ such that

$$s^*(\varphi) \leq \psi \iff \varphi \leq \Pi_s(\psi).$$

# Tripos: quantifiers as adjoints

For any $s : \Gamma \to \Gamma'$, there is a **pHA**-morphism $\Pi_s : \mathcal{T}(\Gamma) \to \mathcal{T}(\Gamma')$ such that

$$s^*(\varphi) \leq \psi \iff \varphi \leq \Pi_s(\psi).$$

Think of $\Pi_s(\varphi)(\overrightarrow{y})$ as

$$\forall \overrightarrow{x}.\ s(\overrightarrow{x}) = \overrightarrow{y} \implies \varphi(\overrightarrow{x}).$$

# Tripos: quantifiers as adjoints

For any $s : \Gamma \to \Gamma'$, there is a **pHA**-morphism $\Pi_s : \mathcal{T}(\Gamma) \to \mathcal{T}(\Gamma')$ such that

$$s^*(\varphi) \leq \psi \iff \varphi \leq \Pi_s(\psi).$$

Think of $\Pi_s(\varphi)(\overrightarrow{y})$ as

$$\forall \overrightarrow{x}.\ s(\overrightarrow{x}) = \overrightarrow{y} \implies \varphi(\overrightarrow{x}).$$

In particular, if $s : \Gamma \times X \to \Gamma$ is a projection, we get the normal $\forall$:

$$\varphi(\overrightarrow{y}, x) \leq \psi(\overrightarrow{y}, x) \iff \varphi(\overrightarrow{y}) \leq \forall x.\psi(\overrightarrow{y}, x)$$

Similar construction for existential quantification.

# Tripos for an evidenced frame: UFam

- Let $\mathcal{EF} = (\Phi, E, \cdot \stackrel{\cdot}{\to} \cdot)$ be an evidenced frame.
- **Idea:** Make a Heyting prealgebra of functions $\Gamma \to \Phi$. $\phi \leq \phi'$ if there is an evidence that works for every $\gamma \in \Gamma$ (uniform families).

# Tripos for an evidenced frame: UFam

- Let $\mathcal{EF} = (\Phi, E, \cdot \xrightarrow{\cdot} \cdot)$ be an evidenced frame.
- **Idea:** Make a Heyting prealgebra of functions $\Gamma \to \Phi$. $\phi \leq \phi'$ if there is an evidence that works for every $\gamma \in \Gamma$ (uniform families).
- Define $\mathcal{T}(\Gamma) = \Gamma \to \Phi$.
- Heyting algebraic structure is given by applying the operations of $\mathcal{EF}$ pointwise.
- Define $\phi \leq \phi' = \exists e. \forall \gamma. \phi(\gamma) \xrightarrow{e} \phi'(\gamma)$.

# Tripos for an evidenced frame: UFam

- Let $\mathcal{EF} = (\Phi, E, \cdot \dot{\rightarrow} \cdot)$ be an evidenced frame.
- **Idea:** Make a Heyting prealgebra of functions $\Gamma \to \Phi$. $\phi \leq \phi'$ if there is an evidence that works for every $\gamma \in \Gamma$ (uniform families).
- Define $\mathcal{T}(\Gamma) = \Gamma \to \Phi$.
- Heyting algebraic structure is given by applying the operations of $\mathcal{EF}$ pointwise.
- Define $\phi \leq \phi' = \exists e. \forall \gamma. \phi(\gamma) \xrightarrow{e} \phi'(\gamma)$.
- Substitution: $s^*(f) = f \circ s$.

# Tripos for an evidenced frame: UFam

- Let $\mathcal{EF} = (\Phi, E, \cdot \xrightarrow{\cdot} \cdot)$ be an evidenced frame.
- **Idea:** Make a Heyting prealgebra of functions $\Gamma \to \Phi$. $\phi \leq \phi'$ if there is an evidence that works for every $\gamma \in \Gamma$ (uniform families).
- Define $\mathcal{T}(\Gamma) = \Gamma \to \Phi$.
- Heyting algebraic structure is given by applying the operations of $\mathcal{EF}$ pointwise.
- Define $\phi \leq \phi' = \exists e. \forall \gamma. \phi(\gamma) \xrightarrow{e} \phi'(\gamma)$.
- Substitution: $s^*(f) = f \circ s$.
- Quantifiers: $\Pi_s(\phi)(\gamma') = \top \supset \{\phi(\gamma) \mid \gamma \in \Gamma \land s(\gamma) = \gamma'\}$.

# Tripos for an evidenced frame: UFam

- Let $\mathcal{EF} = (\Phi, E, \cdot \dot{\rightarrow} \cdot)$ be an evidenced frame.
- **Idea:** Make a Heyting prealgebra of functions $\Gamma \rightarrow \Phi$. $\phi \leq \phi'$ if there is an evidence that works for every $\gamma \in \Gamma$ (uniform families).
- Define $\mathcal{T}(\Gamma) = \Gamma \rightarrow \Phi$.
- Heyting algebraic structure is given by applying the operations of $\mathcal{EF}$ pointwise.
- Define $\phi \leq \phi' = \exists e. \forall \gamma. \phi(\gamma) \xrightarrow{e} \phi'(\gamma)$.
- Substitution: $s^*(f) = f \circ s$.
- Quantifiers: $\Pi_s(\phi)(\gamma') = \top \supset \{\phi(\gamma) \mid \gamma \in \Gamma \wedge s(\gamma) = \gamma'\}$.
- Generic predicate: $\Omega = \Phi$, holds $=$ id, $\chi_\phi = \phi$.

# Tripos for an evidenced frame: UFam

- Let $\mathcal{EF} = (\Phi, E, \cdot \dot{\to} \cdot)$ be an evidenced frame.
- **Idea:** Make a Heyting prealgebra of functions $\Gamma \to \Phi$. $\phi \leq \phi'$ if there is an evidence that works for every $\gamma \in \Gamma$ (uniform families).
- Define $\mathcal{T}(\Gamma) = \Gamma \to \Phi$.
- Heyting algebraic structure is given by applying the operations of $\mathcal{EF}$ pointwise.
- Define $\phi \leq \phi' = \exists e. \forall \gamma. \phi(\gamma) \xrightarrow{e} \phi'(\gamma)$.
- Substitution: $s^*(f) = f \circ s$.
- Quantifiers: $\Pi_s(\phi)(\gamma') = \top \supset \{\phi(\gamma) \mid \gamma \in \Gamma \wedge s(\gamma) = \gamma'\}$.
- Generic predicate: $\Omega = \Phi$, holds $=$ id, $\chi_\phi = \phi$.
- It is also possible to construct an evidenced frame from a tripos.
- This essentially means that any tripos can be described as an evidenced frame.

# Conclusion

**HOL** $\xrightarrow[\text{Syntactic translation}]{}$ **EffHOL** $\xrightarrow[\text{Semantics}]{}$ **Evidenced frame** $\xrightarrow[\text{Uniform families}]{}$ **Tripos**

- We defined two different models of **HOL** that allow for effectful realizers.
- **EffHOL** is a system consisting of an effectful programming language and a logic system with modality.
  - Effects are achieved via a monadic type former.
  - **HOL** theorems are translated to types and specifications, proofs to programs.
- Evidenced frames are defined as a proof relevant ordering.
  - The evidence can be effectful, e.g. computational systems.
- Triposes are category-theoretical models of **HOL**.
  - Propositional logic is implemented through Heyting prealgebras.
  - Predicate logic is implemented through a functor $\mathbf{Set}^{\mathsf{op}} \to \mathbf{pHA}$.

# Questions

# Bonus: translating HOL to EffHOL: complicated example

Consider the following **HOL** proposition:

$$\forall x : ☆.\ \forall_y : ☆.\ (\forall p : ☆ \to ☆.\ x \in p \sqsupset y \in p) \sqsupset \overline{x} \sqsupset \overline{y}$$

The corresponding **EffHOL** type is:

$$\prod X : \star.\ M(\prod Y : \star.\ M((\prod P : \star \to \star. M(P\ X \to M(P\ Y))) \to M(X \to M(Y))))$$

The corresponding **EffHOL** specification is:

$$p \mapsto \cap X : \star.\ \forall x : R_X.\ \langle x_0 \leftarrow p\ X \rangle \cap Y : \star.\ \forall y : R_Y.$$
$$\langle x_1 \leftarrow x_0\ Y \rangle\ \sqcap x_2 : \prod P : \star \to \star.\ M(P\ X \to M(P\ Y)).$$
$$(\cap P : \star \to \star.\ \forall p : \wedge X_0 : \star.\ R_{P\ X_0}(R_{X_0}).\ \langle x_6 \leftarrow x_2\ P \rangle$$
$$\sqcap x_7 : P\ X.\ x_7; x \Subset p\ X \supset \langle x_8 \leftarrow x_6\ x_7 \rangle\ x_8; y_y \Subset p\ Y\big)$$
$$\supset \langle x_3 \leftarrow x_1\ x_2 \rangle\ \sqcap x_4 : X.\ x_4 \Subset x \supset \langle x_5 \leftarrow x_3\ x_4 \rangle\ x_5 \Subset y$$

# Bonus: proof of the soundness theorem

We translate each proof rule to an operation on **EffHOL** programs.

- Id: $[p]$.
- Imp-I: $\lambda x : [\![\varphi_1]\!]^T.\ p$.
- Imp-E: let $x_0 \leftarrow p_0$ in let $x_1 \leftarrow p_1$ in $x_0\ x_1$.
- Uni-I: $\Lambda X : [\![s]\!]^K.\ p$.
- Uni-E: let $x \leftarrow p$ in $x\ [\![t]\!]^t$.
- Rules for comprehensions: do nothing.

## Example

A realizer for $\forall x : \star.\ \forall y : \star.\ (\forall p : \star \to \star.\ x \in p \sqsupset y \in p) \sqsupset \overline{x} \sqsupset \overline{y}$ is

$$\Lambda X : \star.\ \Lambda Y : \star.\ \lambda h_1 : \prod P : \star \to \star.\ M(PX \to M(PY)).\ \lambda h_2 : X.$$
$$\text{let } x_1 \leftarrow h_1\ (\overline{\Lambda} a : \star.\ a) \text{ in let } x_2 \leftarrow [h_2] \text{ in } x_1\ x_2.$$

# Bonus: evidenced frame for a computational system, elaborated

Define $c_1 \cdot c_2 \Downarrow^{\sigma} \phi$ as $\forall c, \sigma' \geq \sigma.\ c_1 \cdot c_2 \wedge c_1 \cdot c_2 \downarrow^{\sigma}_{\sigma'} c \implies (\sigma', c) \in \phi$.
Given a computational system $(C, \Sigma)$ with a separator $\mathcal{S}$, define an evidenced frame $(\Phi, E, \cdot \dot{\rightarrow} \cdot)$ as follows:

$$\begin{aligned}
\Phi &= \{\phi \in \mathcal{P}(\Sigma \times C) \mid \forall \sigma, \sigma', c.\ \sigma' \geq \sigma \\
&\quad \implies (\sigma, c) \in \phi \implies (\sigma', c) \in \phi\} \\
E &= \mathcal{S} \\
\top &= \Sigma \times C \\
\phi_1 \wedge \phi_2 &= \{(\sigma, c) \mid \forall \sigma'.\ \sigma' \geq \sigma \implies \pi_1\ c \Downarrow^{\sigma'} \phi_1\ \wedge\ \pi_2\ c \Downarrow^{\sigma'} \phi_2\} \\
\phi_1 \supset \overrightarrow{\phi} &= \{(\sigma, c) \mid \forall \sigma', c', \phi \in \overrightarrow{\phi}.\ \sigma' \geq \sigma \\
&\quad \implies (\sigma', c') \in \phi_1 \implies c \cdot c' \Downarrow^{\sigma'} \phi\}
\end{aligned}$$