Categorical logical relations for effects and handlers

Jesse Sigal (University of Edinburgh)

CATNIP 6, University of Strathclyde, 28/02/2025

Overview

Overview

Logical Relations

Effects and Handlers

Bringing it all together

What are logical relations?

- Logical relations are a method of proof for type theories.
- A method of defining relations inductively over types.
- Introduced by Plotkin 1973, 1980.
- Able to prove things like:
  - Termination: do your programs stop?
  - Type safety: do your programs keep going?
  - Optimizations: why can I rewrite my program?
  - Representation independence: internals don't matter if you hide them.
  - Security: show the output doesn't depend on secure information.
- There are syntactic, semantic, and mixed approaches, we focus on semantic.

Why are logical relations neat?

- The logical relation interpretation $[\![-]\!]$ maps
  - each context Γ to a binary relation $[\![\Gamma]\!] \subseteq G_1 \times G_2$;
  - each type A to a binary relation $[\![A]\!] \subseteq X_1 \times X_2$; and
  - each program $\Gamma \vdash M : A$ to a pair of functions $[\![M]\!] = (f_1, f_2)$ where $f_i \colon G_i \to X_i$.
- The *fundamental theorem of logical relations* says that given *any* program

  $$\Gamma \vdash M : A$$

  that

  $$\forall(\gamma_1, \gamma_2) \in [\![\Gamma]\!] \,.\, (f_1(\gamma_1), f_2(\gamma_2)) \in [\![A]\!]$$

  or equivalently

  $$(f_1 \times f_2)\left[[\![\Gamma]\!]\right] \subseteq [\![A]\!] \,.$$

- Thus we prove a property about all programs!

What language will we work with?

- We will start with a simple language which has:
  - base types
  - finite products
  - finite coproducts
  - exponentials
- A category $\mathcal{C}$ with these constructions is a bi-cartesian closed category (bi-CCC).
- For each such $\mathcal{C}$, when we choose an object for each base type, we get a completely determined interpretation $[\![-]\!]_{\mathcal{C}}$ which maps
  - each context Γ to an object $[\![\Gamma]\!]_{\mathcal{C}}$;
  - each type A to an object $[\![A]\!]_{\mathcal{C}}$; and
  - each program $\Gamma \vdash M : A$ a morphism $[\![M]\!]_{\mathcal{C}} : [\![\Gamma]\!]_{\mathcal{C}} \to [\![A]\!]_{\mathcal{C}}$.
- Clearly, **Set** is a bi-CCC, and $[\![-]\!]_{\mathbf{Set}}$ is the *standard* semantics.
- We want to prove things about the standard semantics.

Binary logical relations

- The *binary* logical relation interpretation $[\![-]\!]$ maps
  - each context Γ to a binary relation $[\![\Gamma]\!] \subseteq G_1 \times G_2$;
  - each type A to a binary relation $[\![A]\!] \subseteq X_1 \times X_2$; and
  - each program $\Gamma \vdash M : A$ to a pair of functions $[\![M]\!] = (f_1, f_2)$ where $f_i \colon G_i \to X_i$.
- **Important:** f_1 and f_2 are standard interpretations $[\![M]\!]^1_{\mathbf{Set}}$ and $[\![M]\!]^2_{\mathbf{Set}}$ for different base type assignments!
- The *fundamental lemma of logical relations* says that given *any* program

$$\Gamma \vdash M : A$$

that

$$\forall (\gamma_1, \gamma_2) \in [\![\Gamma]\!] \,.\, (f_1(\gamma_1), f_2(\gamma_2)) \in [\![A]\!]$$

or equivalently

$$(f_1 \times f_2)\left[[\![\Gamma]\!]\right] \subseteq [\![A]\!] \,.$$

Unary logical relations

- The *unary* logical relation interpretation $[\![-]\!]$ maps
  - each context Γ to a predicate $[\![\Gamma]\!] \subseteq G$;
  - each type A to a predicate $[\![A]\!] \subseteq X$; and
  - each program $\Gamma \vdash M : A$ to a function $[\![M]\!] = f : G \to X$.
- **Important:** $[\![M]\!]$ is the standard interpretation $[\![M]\!]_{\mathbf{Set}}$!
- The *fundamental theorem of logical relations* says that given *any* program

  $$\Gamma \vdash M : A$$

  that

  $$\forall \gamma \in [\![\Gamma]\!] \,.\, f(\gamma) \in [\![A]\!]$$

  or equivalently

  $$f\left[[\![\Gamma]\!]\right] \subseteq [\![A]\!] \,.$$

- Now for the secret sauce.

Category of predicates

Let **Pred** be the category with

objects: pairs of sets (A, X) such that $A \subseteq X$

morphisms: $f \colon (A, X) \to (B, Y)$ is a function $f \colon X \to Y$ such that $f[A] \subseteq B$.

Pred has finite products, finite coproducts, and exponentials given by

$$\dot{1} = (1, 1) \qquad (A, X) \mathbin{\dot{\times}} (B, Y) = (A \times B, X \times Y)$$
$$\dot{0} = (0, 0) \qquad (A, X) \mathbin{\dot{+}} (B, Y) = (A + B, X + Y)$$
$$(A, X) \mathbin{\dot{\Rightarrow}} (B, Y) = (\{f : f[A] \subseteq B\}, X \Rightarrow Y)$$

We also have a functor $\pi \colon$ **Pred** $\to$ **Set** given by $(A, X) \mapsto X$, $f \mapsto f$.

Key feature: π strictly preserves the bi-cartesian closed (bi-CC) structure

Fundamental theorem of unary logical relations

- We now have two interpretations of our language:
  - $[\![-]\!]_{\mathbf{Set}}$ giving semantics in **Set** and
  - $[\![-]\!]_{\mathbf{Pred}}$ giving semantics in **Pred**

  given completely by the bi-CC structure.
- For a program $\Gamma \vdash M : A$, we have

$$[\![M]\!]_{\mathbf{Set}} : [\![\Gamma]\!]_{\mathbf{Set}} \to [\![A]\!]_{\mathbf{Set}} \qquad [\![M]\!]_{\mathbf{Pred}} : [\![\Gamma]\!]_{\mathbf{Pred}} \to [\![A]\!]_{\mathbf{Pred}}$$

  and because π strictly preserves the bi-CC structure $\pi\,[\![M]\!]_{\mathbf{Pred}} = [\![M]\!]_{\mathbf{Set}}$.
- Thus, where $[\![\Gamma]\!]_{\mathbf{Pred}} = \left(G, [\![\Gamma]\!]_{\mathbf{Set}}\right)$ and $[\![A]\!]_{\mathbf{Pred}} = \left(X, [\![A]\!]_{\mathbf{Set}}\right)$ we get

$$[\![M]\!]_{\mathbf{Set}}\,[G] \subseteq X$$

  which is exactly the fundamental theorem!

Category of relations

We need an analogous category for the binary case. Let $F\colon \mathbf{Set} \times \mathbf{Set} \to \mathbf{Set}$ map $(X, Y) \mapsto X \times Y$. Consider the pullback of categories

$$\begin{array}{ccc} F^*\mathbf{Pred} & \xrightarrow{\quad q \quad} & \mathbf{Pred} \\ {\scriptstyle F^*\pi}\downarrow & & \downarrow{\scriptstyle \pi} \\ \mathbf{Set} \times \mathbf{Set} & \xrightarrow[\quad F \quad]{} & \mathbf{Set} \end{array}$$

Then $F^*\mathbf{Pred}$ has as

objects: pairs $(R, (X, Y))$ where $X, Y \in \mathbf{Set}$ and R is a subset of $X \times Y$

morphisms: $(f_1, f_2)\colon (R, (X_1, X_2)) \to (S, (Y_1, Y_2))$ is a pair of functions $f_i\colon X_i \to Y_i$ such that $(f_1 \times f_2)[R] \subseteq S$

We call this category **BRel**.

Category of relations

BRel has finite products, finite coproducts, and exponentials given by

$$\dot{1} = (1,(1,1)) \quad \dot{0} = (0,(0,0))$$

$$(R,(X_1,X_2)) \mathbin{\dot{\times}} (S,(Y_1,Y_2)) = \left(\mathsf{swap}^{-1}[R \times S], (X_1 \times Y_1, X_2 \times Y_2)\right)$$

$$(R,(X_1,X_2)) \mathbin{\dot{+}} (S,(Y_1,Y_2)) = \left(\iota[R + S], (X_1 + Y_1, X_2 + Y_2)\right)$$

$$(R,(X_1,X_2)) \mathbin{\dot{\Rightarrow}} (S,(Y_1,Y_2)) = \left(\left\{(f_1,f_2) : (f_1 \times f_2)[A] \subseteq B\right\}, (X_1 \Rightarrow Y_1, X_2 \Rightarrow Y_2)\right)$$

where

$$\mathsf{swap}\colon (X_1 \times Y_1) \times (X_2 \times Y_2) \to (X_1 \times X_2) \times (Y_1 \times Y_2)$$

$$\iota\colon (X_1 \times X_2) + (Y_1 \times Y_2) \to (X_1 + Y_1) \times (X_2 + Y_2)$$

Note: we need preimage, direct image, and for exponentials that F is product preserving.

Key feature: $F^*\pi$ strictly preserves the bi-CC structure

Fundamental theorem of binary logical relations

- We now have two interpretations of our language $[\![-]\!]_{\mathbf{Set}\times\mathbf{Set}}$ and $[\![-]\!]_{\mathbf{BRel}}$ given completely by the bi-CC structure.
- Note that $[\![-]\!]_{\mathbf{Set}\times\mathbf{Set}} = ([\![-]\!]^1_{\mathbf{Set}}, [\![-]\!]^2_{\mathbf{Set}})$ for two different base type assignments.
- For a program $\Gamma \vdash M : A$, we have

$$[\![M]\!]_{\mathbf{Set}\times\mathbf{Set}} : [\![\Gamma]\!]_{\mathbf{Set}\times\mathbf{Set}} \to [\![A]\!]_{\mathbf{Set}\times\mathbf{Set}} \qquad [\![M]\!]_{\mathbf{BRel}} : [\![\Gamma]\!]_{\mathbf{BRel}} \to [\![A]\!]_{\mathbf{BRel}}$$

  and $F^*\pi$ strictly preserves the bi-CC structure so $F^*\pi\,[\![M]\!]_{\mathbf{BRel}} = [\![M]\!]_{\mathbf{Set}\times\mathbf{Set}}$.
- Thus, where $[\![\Gamma]\!]_{\mathbf{BRel}} = \left(G, [\![\Gamma]\!]_{\mathbf{Set}\times\mathbf{Set}}\right)$ and $[\![A]\!]_{\mathbf{BRel}} = \left(X, [\![A]\!]_{\mathbf{Set}\times\mathbf{Set}}\right)$ we get

$$\left([\![M]\!]^1_{\mathbf{Set}} \times [\![M]\!]^2_{\mathbf{Set}}\right)[G] \subseteq X$$

  which is exactly the fundamental theorem!

How do we generalize?

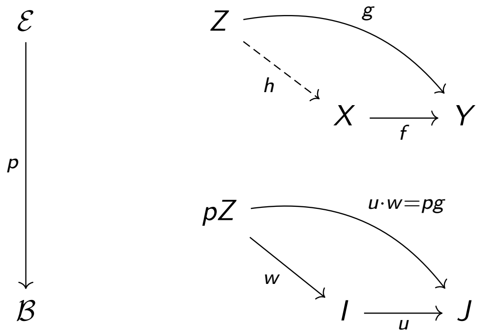
- The key ingredients for unary logical relations were
  - a bi-CCC $\mathcal{C}$;
  - a bi-CCC $\mathcal{E}$; and
  - a functor $p\colon \mathcal{E} \to \mathcal{C}$ which strictly preserved the bi-CC structure.
  - (less important, a form of thinness)
- The key ingredients for deriving binary logical relations from unary ones were
  - for each morphism f in $\mathcal{C}$ a preimage $f^{-1}[-]$ for $\mathcal{E}$;
  - for each morphism f in $\mathcal{C}$ a direct image $f[-]$ for $\mathcal{E}$; and
  - that we pull back along a product preserving functor.
- We want both features, and more in order to support effects and handlers.

Fibrations

- Let $p \colon \mathcal{E} \to \mathcal{C}$ be a functor. We will call $\mathcal{E}$ the *total category* and $\mathcal{C}$ the *base category*.
- We will only work with p's which are faithful.
- $X \in \mathcal{E}$ such that $pX = I \in \mathcal{C}$ is said to be *above* I.
- A morphism f of $\mathcal{E}$ with $pf = u$ of $\mathcal{C}$ is said to be *above* u.
- The subcategory $\mathcal{E}_I$ of $\mathcal{E}$ consisting of the objects above I and morphisms above id_I is called the *fibre category*, or simply *fibre*, over I.
- We will only work with p's such that each $\mathcal{E}_I$ is a partial order.

Cartesian lifts

A morphism $f\colon X \to Y$ in $\mathcal{E}$ is *Cartesian over* $u\colon I \to J$ in $\mathcal{C}$ if $pf = u$ and every $g\colon Z \to Y$ in $\mathcal{E}$ for which one has $pg = u \cdot w$ for some $w\colon pZ \to I$, uniquely determines an $h\colon Z \to X$ in $\mathcal{E}$ above w with $f \cdot h = g$.



For faithful p, any lift is unique, and if it exists write $u\colon X \dot{\to} Y$.

Fibrations, generalized preimage

- When every map in the base category $\mathcal{C}$ has a cartesian lift, we say p is a *fibration*.
- Under our assumptions, these lifts organize into functors.
- For each $u\colon I \to J$ in $\mathcal{C}$, we get a functor

$$u^*\colon \mathcal{E}_J \to \mathcal{E}_I$$

- Fact: $\pi\colon \mathbf{Pred} \to \mathbf{Set}$ is a fibration and for $f\colon X \to Y$,

$$\begin{aligned} f^*\colon \mathbf{Pred}_Y &\to \mathbf{Pred}_X \\ (B, Y) &\mapsto \left(f^{-1}[B], X\right) \end{aligned}$$

Fibrations, generalized direct image

- The important property of direct image is

$$f[A] \subseteq B \iff A \subseteq f^{-1}[B]$$

- Thus, for each $u\colon I \to J$ in $\mathcal{C}$ we want the functor $u^*\colon \mathcal{E}_J \to \mathcal{E}_I$ to have a left adjoint

$$u_*\colon \mathcal{E}_I \to\colon \mathcal{E}_J$$

- When each u^* has a left adjoint, we say p is a *bifibration*.
- Fact: $\pi\colon \mathbf{Pred} \to \mathbf{Set}$ is a bifibration and for $f\colon X \to Y$,

$$\begin{aligned} f_*\colon \mathbf{Pred}_X &\to \mathbf{Pred}_Y \\ (A, X) &\mapsto \big(f[A], Y\big) \end{aligned}$$

Putting it all together

Thus, we want

- a bi-CCC $\mathcal{C}$,
- a bi-CCC $\mathcal{E}$,
- a faithful functor $p\colon \mathcal{E} \to \mathcal{C}$,
- p to strictly preserve the bi-CC structure,
- the fibre categories to be partial orders,
- p to be a bifibration, and
- (later) the fibre categories to have small products.

This is the definition of a fibration for logical relations of Katsumata 2013. FFLRs subsume sconing and Kripke logical relations with varying arity.

Overview

Logical Relations

Effects and Handlers

Bringing it all together

What are effects and handlers?

- Real life programs need side effects.
- Side effects are often modelled with monads, but monads don't compose!
- Effects and handlers are a modular and composable way for users to define their own effects.
- Specifically, effectful operations have no meaning except when given one by user defined handlers.
- This is achieved with a special free monad for which handlers induce monad algebras, see Forster et al. 2019.
- Real languages like WebAssembly and OCaml have effects and handlers!

Kleisli categories for effects

- A standard model for side-effects in programming languages is a cartesian closed category $\mathcal{C}$ equipped with a strong monad $\mathcal{T} = (T, \eta, \mu, \mathrm{st})$.
- Recall that a strength for a functor $F\colon \mathcal{C} \to \mathcal{C}$ is a map $\mathrm{st}\colon X \times FY \to F(X \times Y)$, and when F is a monad, some compatibility conditions.
- The semantics $[\![-]\!]_{\mathcal{T}}$ then assigns to each program $\Gamma \vdash M : A$ a morphism

$$[\![M]\!]_{\mathcal{T}} : [\![\Gamma]\!]_{\mathcal{T}} \to T\,[\![A]\!]_{\mathcal{T}}$$

  in the Kleisli category $\mathcal{C}_{\mathcal{T}}$.
- If we want to add a built-in effectful operation $\mathrm{op}\colon A \to B$ to the language, we choose a map

$$[\![A]\!]_{\mathcal{T}} \to T\,[\![B]\!]_{\mathcal{T}}$$

Algebraic effects

- An *algebraic operation* α from A to B for a strong monad $\mathcal{T}$ is a natural transformation
$$\alpha_X \colon (B \Rightarrow TX) \to (A \Rightarrow TX)$$
which respects η and μ.
- Algebraic operations are in bijection with Kleisli morphism given (morally) by
$$\begin{aligned}(B \Rightarrow TX) \to (A \Rightarrow TX) &\cong A \to TB \\ \alpha &\mapsto \alpha_B\,(\eta_B) \\ \lambda g.(g \cdot_{\mathcal{C}_{\mathcal{T}}} f) &\mapsfrom f\end{aligned}$$

A special functor

- Suppose we have n algebraic operations $\mathrm{op}_i \colon A_i \to B_i$ we want to support. Then we have

$$\begin{aligned} & \text{for } i = 1, \dots, n : (B_i \Rightarrow TX) \to (A_i \Rightarrow TX) \\ \cong\ & \text{for } i = 1, \dots, n : A_i \times (B_i \Rightarrow TX) \to TX \\ \cong\ & \sum_{i=1}^{n} A_i \times (B_i \Rightarrow TX) \to TX \end{aligned}$$

- Thus, if we define a functor $F \colon \mathcal{C} \to \mathcal{C}$ as

$$FY := \sum_{i=1}^{n} A_i \times (B_i \Rightarrow Y)$$

then we are asking for an F-algebra structure on TX

$$F(TX) \to TX$$

natural in X.

Effect functor

- We call a functor

$$FY = \sum_{i=1}^{n} A_i \times (B_i \Rightarrow Y)$$

  an *effect functor*.
- A monad $\mathcal{T}$ *supports* F when there is a natural transformation $\varphi \colon FT \to T$ so each component is an F-algebra.
- How can we find such a monad for an arbitrary F?

Free algebras

A *free F-algebra* on an object A in $\mathcal{C}$ is an algebra $\varphi_A \colon FA^\sharp \to A^\sharp$ such that for every $\beta \colon FB \to B$ and every morphism $f \colon A \to B$ in $\mathcal{C}$, there exists a unique homomorphism $\bar{f} \colon A^\sharp \to B$ extending f.

$$\begin{array}{ccccc} FA^\sharp & \xrightarrow{\varphi_A} & A^\sharp & \xleftarrow{\eta_A} & A \\ {\scriptstyle F\bar{f}}\downarrow & & {\scriptstyle \bar{f}}\downarrow & \swarrow {\scriptstyle f} & \\ FB & \xrightarrow[\beta]{} & B & & \end{array}$$

Thus, a free algebra (if it exists) is a minimal way to equip A with an F-algebra structure.

Free monads

- If every object has a free F-algebra, these coalesce into a monad $\mathcal{T} = (T, \eta, \mu)$ on $\mathcal{C}$ by $TA = A^\sharp$.
- This is the *free algebraic monad*, and there is a natural transformation $\varphi\colon FT \to T$ and so is component wise F-algebras.
- There is a strictly weaker definition of *free monad*, but we require the stronger version.
- Importantly, there is an equivalence of categories between F-algebras and $\mathcal{T}$-algebras.
- Finally, when F is strong, so is $\mathcal{T}$.

Handlers

- Given an effect functor F, the programming language construct of a *handler* generates an F-algebra.
- Thus, we choose our semantics to work with the free monad $\mathcal{T}$ on F, we get a $\mathcal{T}$-algebra.
- Therefore, users can “escape” from the monad by giving their chosen interpretation of the supported effects.

Semantics

The basic semantics for effects and handlers is completely determined by

- a bi-CCC $\mathcal{C}$ such that free monads $\mathcal{T}$ of effect functors exist and
- the free-forgetful adjunction

$$\mathcal{C} \underset{|-|}{\overset{\perp}{\rightleftarrows}} \mathbf{Alg}\ \mathcal{T}$$

The semantics $[\![-]\!]_{\mathcal{C}}$ then assigns to each program $\Gamma \vdash M : C$ a morphism in $\mathcal{C}$

$$[\![M]\!]_{\mathcal{C}} : [\![\Gamma]\!]_{\mathcal{C}} \to \left|[\![C]\!]_{\mathcal{C}}\right|$$

Overview

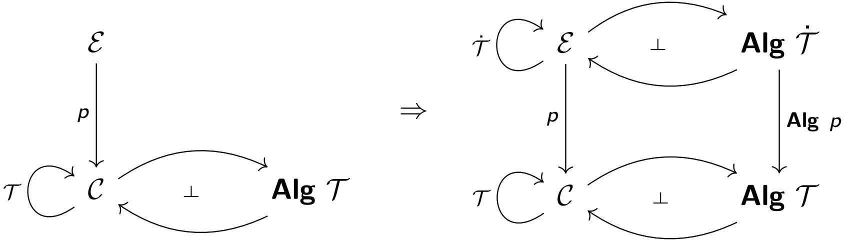
Logical Relations

Effects and Handlers

Bringing it all together

Lifting monads

We currently have the left hand side. We must find a strong monad $\dot{\mathcal{T}}$ on $\mathcal{E}$ which is a *lift* of $\mathcal{T}$, resulting in the right hand side.



$\dot{\mathcal{T}} = (\dot{T}, \dot{\eta}, \dot{\mu}, \dot{\mathsf{st}})$ is a lift when

$$p(\dot{T}X) = T(pX), \quad p\dot{\eta}_X = \eta_{pX}, \quad p\dot{\mu}_X = \mu_{pX}, \quad p\dot{\mathsf{st}}_{X,Y} = \mathsf{st}_{pX,pY}$$

and such a $\dot{\mathcal{T}}$ gives us the right hand side.

The correct lifting

- In general, there are many liftings of a monad $\mathcal{T}$.
- We also have a stronger requirement. An effect functor on $\mathcal{C}$

$$FC = \sum_{i=1}^{n} A_i \times (B_i \Rightarrow C)$$

and choices of $pX_i = A_i, pY_i = B_i$ induces one on $\mathcal{E}$

$$\dot{F}Z = \dot{\sum_{i=1}^{n}} X_i \mathbin{\dot{\times}} (Y_i \mathbin{\dot{\Rightarrow}} Z)$$

and so $p\dot{F} = Fp$.

- We need $\dot{\mathcal{T}}$ to be the free algebra monad for $\dot{F}$ for our semantics.

Free algebriac lift

The answer is the *free algebraic lift* of Kammar and McDermott 2018.

Let $\{\alpha_i \colon (B_i \Rightarrow T-) \to (A_i \Rightarrow T-)\}_{1 \leq i \leq n}$ be a set of algebraic operations of $\mathcal{T}$ and $Y_i, Z_i \in \mathcal{E}$ above $A_i, B_i \in \mathcal{C}$ respectively.

For each object $X \in \mathcal{E}$, define $\mathcal{R}X$ as the set of all $X' \in \mathcal{E}_{T(pX)}$ such that:

- The unit respects X', i.e. $\eta \colon X \dot{\to} X'$.
- Each algebraic operation respects X' for the given lift, i.e. $\alpha_i \colon (Z_i \dot{\Rightarrow} X') \dot{\to} (Y_i \dot{\Rightarrow} X')$

Define $\dot{T}X \coloneqq \bigwedge \mathcal{R}X$, i.e. $\dot{T}X$ is the least element of $\mathcal{R}X$ ($\bigwedge$ product in $\mathcal{E}_{T(pX)}$).

Then $\dot{T}$ is part of a monad lift $\dot{\mathcal{T}}$. Furthermore, each algebraic operation α_i lifts to an algebraic operation $\dot{\alpha}_i$.

Free for free

Theorem

Let $p\colon \mathcal{E} \to \mathcal{C}$ be an FFLR, F an effect functor, $\dot{F}$ it's lift, and $\mathcal{T}$ the free algebra monad for F. Then the free algebraic lift $\dot{\mathcal{T}}$ with respect to the operations supported by F is the free algebra monad for $\dot{F}$

Sketch.

The monad $\mathcal{T}$ is "exactly" made up of the algebraic operations it supports, and so if we take a lift which respects them, the lift respects this "exactness". □

Note: the proof makes use of all pieces of the FFLR definition.

Fundamental theorem of logical relations

- We now have two interpretations of our effects an handler language:
  - $[\![-]\!]_{\mathcal{C}}$ giving semantics in $\mathcal{C}$ and
  - $[\![-]\!]_{\mathcal{E}}$ giving semantics in $\mathcal{E}$

  given completely by the bi-CC structure and existence of free algebra monads for effect functors.
- For a program $\Gamma \vdash M : C$, we have

$$[\![M]\!]_{\mathcal{C}} : [\![\Gamma]\!]_{\mathcal{C}} \to \left|[\![C]\!]_{\mathcal{C}}\right| \quad [\![M]\!]_{\mathcal{E}} : [\![\Gamma]\!]_{\mathcal{E}} \to \left|[\![C]\!]_{\mathcal{E}}\right|$$

- p and **Alg** p strictly preserve the used structure, and so we have

$$p\,[\![M]\!]_{\mathcal{E}} = [\![M]\!]_{\mathcal{C}}$$

  which is exactly a generalized fundamental theorem.

Conclusion

- Logical relations let us prove powerful theorems about languages.
- Fibrations for logical relations provide a categorical extension of logical relations, including constructing new from old.
- Effects and handlers let one program with certain free monads.
- FFLRs let us do logical relations for effects and handlers via the free algebraic lift.

References I

- Forster, Yannick et al. (2019). "On the expressive power of user-defined effects: Effect handlers, monadic reflection, delimited control". In: *Journal of Functional Programming* 29. Publisher: Cambridge University Press. ISSN: 0956-7968, 1469-7653. DOI: 10.1017/S0956796819000121. URL: https://www.cambridge.org/core/journals/journal-of-functional-programming/article/on-the-expressive-power-of-userdefined-effects-effect-handlers-monadic-reflection-delimited-control/3FFAA9AD05B58A1467E411F80EE4E076 (visited on 04/17/2020).
- Kammar, Ohad and Dylan McDermott (Dec. 1, 2018). "Factorisation Systems for Logical Relations and Monadic Lifting in Type-and-effect System Semantics". In: *Electronic Notes in Theoretical Computer Science* 341. Publisher: Elsevier, pp. 239–260. ISSN: 1571-0661. DOI: 10.1016/j.entcs.2018.11.012. URL: https://www.sciencedirect.com/science/article/pii/S1571066118300938 (visited on 10/25/2023).

- Katsumata, Shin-ya (Jan. 1, 2013). "Relating computational effects by $\top\top$-lifting". In: *Information and Computation*. 38th International Colloquium on Automata, Languages and Programming (ICALP 2011) 222, pp. 228–246. ISSN: 0890-5401. DOI: 10.1016/j.ic.2012.10.014. URL: https://www.sciencedirect.com/science/article/pii/S0890540112001551 (visited on 09/05/2022).
- Plotkin, Gordon D. (Oct. 1973). *λ-definability and logical relations*. SAI-RM-4, p. 20. URL: https://homepages.inf.ed.ac.uk/gdp/publications/logical_relations_1973.pdf (visited on 02/26/2025).
- — (1980). "λ-definability in the full type hierarchy". In: *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*. Ed. by Haskell B. Curry, James R. Hindley, and Jonathan P. Seldin. New York, NY, USA: Academic Press, pp. 363–373. ISBN: 0-12-349050-2. URL: https://homepages.inf.ed.ac.uk/gdp/publications/Lambda_Definability.pdf (visited on 02/26/2025).