Denotational Semantics of Gradual Typing using Synthetic Guarded Domain Theory

Eric Giovannini
Electrical Engineering and Computer Science
University of Michigan
USA
ericgio@umich.edu

Tingting Ding
Electrical Engineering and Computer Science
University of Michigan
USA
tingtind@umich.edu

Max S. New
Electrical Engineering and Computer Science
University of Michigan
USA
maxsnew@umich.edu

ABSTRACT

Gradually typed programming languages, which allow for soundly mixing static and dynamically typed programming styles, present a strong challenge for metatheorists. Even the simplest sound gradually typed languages feature at least recursion and errors, with realistic languages featuring furthermore runtime allocation of memory locations and dynamic type tags. Further, the desired metatheoretic properties of gradually typed languages have become increasingly sophisticated: validity of type based equational reasoning as well as the relational property known as the gradual guarantee or graduality. Many recent works have tackled verifying these properties, but the resulting mathematical developments are highly repetitive and tedious, with few reusable theorems persisting across different developments.

In this work, we present a new denotational account of gradual typing semantics developed using guarded domain theory. Guarded domain theory combines the expressive power of step-indexed logical relations for modeling recursive features with the modularity and reusability of denotational semantics. Further, recent extensions to cubical Agda mean that synthetic guarded domain theory is readily mechanized in a proof assistant. We demonstrate the feasibility of this approach with a model of gradually typed lambda calculus and prove the validity of beta-eta equality and the graduality theorem for the denotational model. This model should provide the basis for a reusable mathematical theory of gradually typed program semantics.

ACM Reference Format:
Eric Giovannini, Tingting Ding, and Max S. New. 2018. Denotational Semantics of Gradual Typing using Synthetic Guarded Domain Theory. In *Woodstock '18: ACM Symposium on Neural Gaze Detection, June 03–05, 2018, Woodstock, NY*. ACM, New York, NY, USA, 19 pages. https://doi.org/10.1145/1122445.1122456

Woodstock '18, June 03–05, 2018, Woodstock, NY

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

1.1 Gradual Typing and Graduality

In modern programming language design, there is a tension between *static* typing and *dynamic* typing disciplines. With static typing, the code is type-checked at compile time, while in dynamic typing, the type checking is deferred to run-time. Both approaches have benefits: with static typing, the programmer is assured that if the program passes the type-checker, their program is free of type errors, and moreover, soundness of the equational theory implies that program optimizations are valid. Meanwhile, dynamic typing allows the programmer to rapidly prototype their application code without needing to commit to fixed type signatures for their functions. Most languages choose between static or dynamic typing and as a result, programmers that initially write their code in a dynamically typed language in order to benefit from faster prototyping and development time need to rewrite the some or all of their codebase in a static language if they would like to receive the benefits of static typing once their codebase has matured.

Gradually-typed languages [30, 33] seek to resolve this tension by allowing for both static and dynamic typing disciplines to be used in the same codebase, and by supporting smooth interoperability between statically-typed and dynamically-typed code. This flexibility allows programmers to begin their projects in a dynamic style and enjoy the benefits of dynamic typing related to rapid prototyping and easy modification while the codebase "solidifies". Over time, as parts of the code become more mature and the programmer is more certain of what the types should be, the code can be *gradually* migrated to a statically typed style without needing to rewrite the project in a completely different language.

In order for this to work as expected, gradually-typed languages should allow for different parts of the codebase to be in different places along the spectrum from dynamic to static, and allow for those different parts to interact with one another. Moreover, gradually-typed languages should support the smooth migration from dynamic typing to static typing, in that the programmer can initially leave off the typing annotations and provide them later without altering the meaning of the program. Furthermore, the parts of the program that are written in a dynamic style should soundly interoperate with the parts that are written in a static style. That is, the interaction between the static and dynamic components of the codebase should preserve the guarantees made by the static types. In particular, while statically-typed code can error at runtime in a gradually-typed language, such an error can always be traced

back to a dynamically-typed term that violated the typing contract imposed by statically typed code.

One of the fundamental theorems for gradually typed languages is *dynamic gradual guarantee*, originally defined by Siek, Vitousek, Cimini, and Boyland [31], also called *graduality* [24], by analogy with parametricity. Informally, graduality says that going from a dynamic to static style should only allow for the introduction of static or dynamic type errors, and not otherwise change the meaning of the program. This is a way to capture programmer intuition that increasing type precision corresponds to a generalized form assertions at runtime only, and so they can trust that prior observed behavior of their dynamically typed code will remain unchanged as long as it satisfies the new stricter typing discipline.

Additionally, gradually typed languages should offer some of the benefits of static typing. While classical type soundness, that well-typed programs are free from runtime errors, is not compatible with runtime type errors, we can instead focus on *type-based reasoning*. For instance, while dynamically typed λ calculi only satisfy β equality for their type formers, statically typed λ calculi additionally satisfy type-dependent η properties that ensure that functions are determined by their behavior under application and that pattern matching on data types is safe and exhaustive.

Our goal in this work is to provide a reusable semantic framework for gradually typed languages that can be used to prove the aforementioned properties: graduality and type-based reasoning.

1.2 Limitations of Prior Work

We give an overview of current approaches to proving graduality of languages and why they do not meet our criteria of a reusable semantic framework.

1.2.1 From Static to Gradual. Current approaches to constructing languages that satisfy the graduality property include the methods of Abstracting Gradual Typing [14] and the formal tools of the Gradualizer [7]. These allow the language developer to start with a statically typed language and derive a gradually typed language that satisfies the gradual guarantee. The main downside to these approaches lies in their inflexibility: since the process in entirely mechanical, the language designer must adhere to the predefined framework. Many gradually typed languages do not fit into either framework, e.g., Typed Racket [33, 34] and the semantics produced is not always the desired one. Furthermore, while these frameworks do prove graduality of the resulting languages, they do not show the correctness of the equational theory, which is equally important to sound gradual typing.

1.2.2 Double Categorical Semantics. New and Licata [27] developed an axiomatic account of the graduality relation on a call-by-name cast calculus terms and showed that the graduality proof could be modeled using semantics in certain kinds of *double categories*, categories internal to the category of categories. A double category extends a category with a second notion of morphism, often a notion of "relation" to be paired with the notion of functional morphism, as well as a notion of functional morphisms preserving relations. In gradual typing the notion of relation models type precision and the squares model the term precision relation. This approach was influenced by the semantics of parametricity using reflexive graph categories [10, 16, 19]: reflexive graph categories are essentially double categories without a notion of relational composition. In addition to capturing the notions of type and term precision, the double categorical approach allows for a *universal property* for casts: upcasts are the *universal* way to turn a relation arrow into a function in a forward direction and downcasts are the dual universal arrow. Later, New, Licata and Ahmed [28] extended this axiomatic treatment from call-by-name to call-by-value as well by giving an axiomatic theory of type and term precision in call-by-push-value. This left implicit any connection to a "double call-by-push-value", which we make explicit in Section 4.

With this notion of abstract categorical model in hand, denotational semantics is then the work of constructing concrete models that exhibit the categorical construction. New and Licata [27] present such a model using categories of ω-CPOs, and this model was extended by Lennon-Bertrand, Maillard, Tabareau and Tanter to prove graduality of a gradual dependently typed calculus CastCIC$^{\mathcal{G}}$. This domain-theoretic approach meets our criteria of being a semantic framework for proving graduality, but suffers from the limitations of classical domain theory: the inability to model viciously self-referential structures such as higher-order extensible state and similar features such as runtime-extensible dynamic types. Since these features are quite common in dynamically typed languages, we seek a new denotational framework that can model these type system features.

The standard alternative to domain theory that scales to essentially arbitrary self-referential definitions is *step-indexing* or its synthetic form of *guarded recursion*. A series of works [24, 26, 28] developed step-indexed logical relations models of gradually typed languages based on operational semantics. Unlike classical domain theory, such step-indexed techniques are capable of modeling higher-order store and runtime-extensible dynamic types [1, 2, 23, 26]. However, their proof developments are highly repetitive and technical, with each development formulating a logical relation from first-principles and proving many of the same tedious lemmas without reusable mathematical abstractions. Our goal in the current work is to extract these reusable mathematical principles from these explicit step-indexed to make formalization of realistic gradual languages tractible.

An alternative approach, which we investigate in this paper, is provided by *synthetic guarded domain theory*[5]. The techniques of synthetic guarded domain theory allow us to internalize the step-indexed reasoning normally required in logical relations proofs of graduality, ultimately allowing us to specify the logical relation in a manner that looks nearly identical to a typical, non-step-indexed logical relation. In fact, guarded domain theory goes further, allowing us to define step-indexed *denotational semantics* not just step-indexed relations, just as easily as constructing an ordinary set-theoretic semantics.

In this paper, we develop an adequate denotational semantics that satisfies graduality and soundness of the equational theory of cast calculi using the techniques of SGDT. Our longer-term goal is to mechanize these proofs in a reusable way in the Guarded Cubical Agda proof assistant, thereby providing a framework to use to more easily and conveniently prove that existing languages satisfy graduality and have sound equational theories. Moreover,

the aim is for designers of new languages to utilize the framework to facilitate the design of new provably-correct gradually-typed languages with more complex features.

1.3 Contributions

The main contribution of this work is a categorical and denotational semantics for gradually typed langauges that models not just the term language but the graduality property as well.

(1) First, we give a simple abstract categorical model of GTT using CBPV double categories.
(2) Next, we modify this semantics to develop reflexive graph- and double categorical models that abstract over the details of step-indexed models, and provide a method for constructing such models.
(3) We instantiate the abstract construction to provide a concrete semantics in informal guarded type theory.
(4) We prove that the resulting denotational model is *adequate* for the graduality property: a closed term precision $M \sqsubseteq N : \mathsf{Nat}$ has the expected semantics, that M errors or M and N have the same extensional behavior.

2 BACKGROUND ON GUARDED DOMAIN THEORY

One way to avoid the tedious reasoning associated with step-indexing is to work axiomatically inside of a logical system that can reason about non-well-founded recursive constructions while abstracting away the specific details of step-indexing required if we were working analytically. The system that proves useful for this purpose is called *synthetic guarded domain theory*, or SGDT for short. We provide a brief overview here, but more details can be found in [5].

SGDT offers a synthetic approach to domain theory that allows for guarded recursion to be expressed syntactically via a type constructor $\triangleright : \mathsf{Type} \to \mathsf{Type}$ (pronounced "later"). The use of a modality to express guarded recursion was introduced by Nakano [22]. Given a type A, the type $\triangleright A$ represents an element of type A that is available one time step later. There is an operator $\mathsf{next} : A \to \triangleright A$ that "delays" an element available now to make it available later. We will use a tilde to denote a term of type $\triangleright A$, e.g., $\tilde{M}$.

There is a *guarded fixpoint* operator

$$\mathsf{fix} : \forall T, (\triangleright T \to T) \to T.$$

That is, to construct a term of type T, it suffices to assume that we have access to such a term "later" and use that to help us build a term "now". This operator satisfies the axiom that $\mathsf{fix} f = f(\mathsf{next}(\mathsf{fix} f))$. In particular, this axiom applies to propositions $P : \mathsf{Prop}$; proving a statement in this manner is known as Löb-induction.

The operators $\triangleright$, next, and fix described above can be indexed by objects called *clocks*. A clock serves as a reference relative to which steps are counted. For instance, given a clock k and type T, the type $\triangleright^k T$ represents a value of type T one unit of time in the future according to clock k. If we only ever had one clock, then we would not need to bother defining this notion. However, the notion of *clock quantification* is crucial for encoding coinductive types using guarded recursion, an idea first introduced by Atkey and McBride [3].

Most of the developments in this paper will take place in the context of a single clock k, but later on, we will need to make use of clock quantification.

2.1 Ticked Cubical Type Theory

Ticked Cubical Type Theory [21] is an extension of Cubical Type Theory [8] in which there is an additional sort called *ticks*. Given a clock k, a tick $t : \mathsf{tick}\, k$ serves as evidence that one unit of time has passed according to the clock k. The type $\triangleright A$ is represented as a function from ticks of a clock k to A. The type A is allowed to depend on t, in which case we write $\triangleright^k_t A$ to emphasize the dependence.

The rules for tick abstraction and application are similar to those of dependent Π types. In particular, if we have a term M of type $\triangleright^k A$, and we have available in the context a tick $t' : \mathsf{tick}\, k$, then we can apply the tick to M to get a term $M[t'] : A[t'/t]$. We will also write tick application as M_t. Conversely, if in a context $\Gamma, t : \mathsf{tick}\, k$ we have that M has type A, then in context Γ we have $\lambda t.M$ has type $\triangleright A$.

3 SYNTACTIC THEORY OF GRADUALLY TYPED LAMBDA CALCULUS

Here we give an overview of a fairly standard cast calculus for gradual typing along with its (in-)equational theory that capture our desired notion of type-based reasoning and graduality. The main departure from prior work is our explicit treatment of type precision derivations and an equational theory of those derivations.

We give the basic syntax and select typing rules in Figure 1. We include a dynamic type, a type of numbers, the call-by-value function type $A \rightharpoonup A'$ and products. We include a syntax for *type precision* derivations $c : A \sqsubseteq A'$, the typing is given in Figure 2. Any type precision derivation $c : A \sqsubseteq A'$ induces a pair of casts, the upcast $\langle : \nwarrow c \rangle A \rightharpoonup A'$ and the downcast $\langle c \swarrow : \rangle A' \rightharpoonup A$. The syntactic intuition is that c is a proof that A is "less dynamic" than A'. Semantically, this gives us coercions back and forth where the upcast is (to a first-order) a pure function whereas the downcast can fail. These casts are inserted automatically in an elaboration from a surface language. In this work, we are focused on semantic aspects and so elide these standard details. The syntax of precision derivations includes reflexivity $r(A)$ and transitivity cc' as well as monotonicity $c \rightharpoonup c'$ and $c \times c'$ that are *covariant* in all arguments and finally generators $\mathrm{Inj}_{\mathbb{N}}, \mathrm{Inj}_{\to}, \mathrm{Inj}_{\times}$ that correspond to the type tags of our dynamic type. We additionally impose an equational theory $c \equiv c'$ on the derivations that implies that the corresponding casts are weakly bisimilar in the semantics. We impose category axioms for the reflexivity and transitivity and functoriality for the monotonicity rules. We note the following two admissible principles: any two derivations $c, c' : A \sqsubseteq A'$ of the same fact are equivalent $c \equiv c'$ and for any A, there is a derivation $\mathrm{dyn}(A) : A \sqsubseteq ?$. That is, ? is the "most dynamic" type.

Next, we consider the axiomatic (in)equational reasoning principles for terms: $\beta\eta$ equality and term precision in Figure 3. We include standard CBV $\beta\eta$ rules for function and product types, as well as equations stating that casts are given functorially. Next, we have *term* precision, an extension of type precision to terms. The form of the term precision rule is $\Delta \vdash M \sqsubseteq M' : c$ where Δ is a

$$\begin{array}{rcl} \text{Types } A & ::= & \text{Nat} \mid ? \mid A \rightharpoonup A' \mid A \times A' \\ \text{Type Precision } c & ::= & r(A) \mid \text{Inj}_{\rightarrow} \mid \text{Inj}_{\mathbb{N}} \mid \text{Inj}_{\times} \mid c \rightharpoonup c' \mid c \times c' \\ \text{Values } V & ::= & \text{up}\, c\, V \mid \text{zro} \mid \text{suc}\, V \mid \lambda x.M \mid (V, V') \\ \text{Terms } M, N & ::= & \mathfrak{U} \mid \text{up}\, c\, M \mid \text{dn}\, c\, M \mid \text{zro} \mid \text{suc}\, M \mid \lambda x.M \\ & & \mid M\, N \mid (M, N) \mid \text{let}\, (x, y) = M \,\text{in}\, N \\ \text{Contexts } \Gamma & ::= & \cdot \mid \Gamma, x : A \\ \text{Ctx Precision } \Delta & ::= & \cdot \mid \Delta, x : c \end{array}$$

$$\frac{\Gamma \vdash M : A \qquad c : A \sqsubseteq A'}{\Gamma \vdash \text{up}\, c\, M : A'} \qquad \frac{\Gamma \vdash N : A' \qquad c : A \sqsubseteq A'}{\Gamma \vdash \text{dn}\, c\, NA}$$

$$\Gamma \vdash \mathfrak{U} : A$$

Figure 1: GTLC Cast Calculus Syntax

$$r(A) : A \sqsubseteq A \qquad \frac{c : A \sqsubseteq A' \qquad c' : A' \sqsubseteq A''}{cc' : A \sqsubseteq A''}$$

$$\text{Inj}_{\rightarrow} : D \rightharpoonup D \sqsubseteq D \qquad \text{Inj}_{\mathbb{N}} : \text{Nat} \sqsubseteq D \qquad \text{Inj}_{\rightarrow} : D \times D \sqsubseteq D$$

$$\frac{c_i : A_i \sqsubseteq A_i' \qquad c_o : A_o \sqsubseteq A_o'}{c_i \rightharpoonup c_o : (A_i \rightharpoonup A_o) \sqsubseteq (A_i' \rightharpoonup A_o')}$$

$$\frac{c_1 : A_1 \sqsubseteq A_1' \qquad c_2 : A_2 \sqsubseteq A_2'}{c_1 \times c_2 : (A_1 \times A_2) \sqsubseteq (A_1' \times A_2')} \qquad r(A)c \equiv c \qquad c \equiv cr(A')$$

$$c(c'c'') \equiv (cc')c'' \qquad r(A_i \rightharpoonup A_o) \equiv r(A_i) \rightharpoonup r(A_o)$$

$$r(A_1 \times A_2) \equiv r(A_1) \times r(A_2)$$

$$(c_i \rightharpoonup c_o)(c_i' \rightharpoonup c_o') \equiv (c_i c_i' \rightharpoonup c_o c_o')$$

$$(c_1 \times c_2)(c_1' \times c_2') \equiv (c_1 c_1' \times c_2 c_2')$$

Figure 2: Type Precision Derivations and equational theory

context where variables are assigned to type precision derivations. The judgment is only well formed when every use of $x : c'$ for $c : A \sqsubseteq A'$ is used with type A in M and A' in M' and similarly the output types match c. We elide the congruence rules for every type constructor, e.g., that $M \sqsubseteq M'$ and $N \sqsubseteq N'$ that $M\, N \sqsubseteq M'\, N'$. With such congruence rules, reflexivity $M \sqsubseteq M$ is admissible. Transitivity, on the other hand, is intentionally not taken as a primitive rule, matching the original formulation of the dynamic gradual guarantee. We include a rule that says that equivalent type precision derivations $c \equiv c'$ are equivalent for the purposes of term precision. The next rule is the *retraction* principle, which states that a downcast after an upcast is equivalent to doing nothing at all, since intuitively the upcasted value should already satisfy the type. Here $\sqsupseteq\sqsubseteq$ means we require each is $\sqsubseteq$ the other, with reflexivity precision derivations. Finally, we include 4 rules for reasoning about casts. Intuitively these say that the upcast is a kind of *least upper bound* and dually that the downcast is a *greatest lower bound*. These principles have been shown in prior work to imply that the

$$(\lambda x.M)(V) = M[V/x] \qquad (V : A \rightharpoonup A') = \lambda x.V\, x$$

$$\text{let}\, (x, y) = (V, V') \,\text{in}\, N = N[V/x, V'/y]$$

$$M[V : A \times A'/p] = \text{let}\, (x, y) = p \,\text{in}\, M[(x, y)/p]$$

$$\text{up}\, (r(A))\, M = M \qquad \text{up}\, c'\, \text{up}\, c\, M = \text{up}\, cc'\, M \qquad \text{dn}\, (r(A))\, M = M$$

$$\text{dn}\, c\, \text{dn}\, c'\, M = \text{dn}\, cc'\, M \qquad \frac{\Delta \vdash M \sqsubseteq M' : c \qquad c \equiv c'}{\Delta \vdash M \sqsubseteq M' : c'}$$

$$\Delta \vdash \mathfrak{U} \sqsubseteq M : c \qquad \text{dn}\, c\, \text{up}\, c\, M \sqsupseteq\sqsubseteq M \qquad \frac{M \sqsubseteq M' : cc_r}{\text{up}\, c\, M \sqsubseteq M' : c_r}\ \textsc{UpL}$$

$$\frac{M \sqsubseteq M' : c_l}{M \sqsubseteq \text{up}\, c\, M' : c_l c}\ \textsc{UpR} \qquad \frac{M \sqsubseteq M' : c_r}{\text{dn}\, c\, M \sqsubseteq M' : cc_r}\ \textsc{DnL} \qquad \frac{M \sqsubseteq M' : c_l c}{M \sqsubseteq \text{dn}\, c\, M' : c_l}\ \textsc{DnR}$$

Figure 3: Equality and Term Precision Rules (Selected)

behavior of functorial lifts such as $c \rightharpoonup c'$ and $c \times c'$ are given by functorial actions [27, 28].

In the remainder of this article, we seek to develop a compositional semantics of this calculus that is *adequate* for graduality. This means we should have a semantic relation $\sqsubseteq$ such that if $M \sqsubseteq M'$ then $[\![M]\!] \sqsubseteq [\![M']\!]$ satisfying the following properties for closed terms of type Nat:

(1) $[\![n]\!] \sqsupseteq\sqsubseteq [\![n']\!]$ if and only if $n = n'$ for natural numbers n, n'.
(2) $[\![n]\!], [\![\mathfrak{U}]\!], [\![\Omega]\!]$ are all different (not $\sqsupseteq\sqsubseteq$), where Ω is a diverging term.
(3) If $[\![M]\!] \sqsubseteq [\![M']\!]$ then
   - If $[\![M]\!] \sqsupseteq\sqsubseteq [\![n]\!]$ then $[\![M']\!] \sqsupseteq\sqsubseteq [\![n]\!]$
   - If $[\![M']\!] \sqsupseteq\sqsubseteq [\![\mathfrak{U}]\!]$ then $[\![M]\!] \sqsupseteq\sqsubseteq [\![\mathfrak{U}]\!]$
   - If $[\![M']\!] \sqsupseteq\sqsubseteq [\![n]\!]$ then $[\![M]\!] \sqsupseteq\sqsubseteq [\![n]\!]$ or $[\![M]\!] \sqsupseteq\sqsubseteq [\![\mathfrak{U}]\!]$

The first two conditions ensure that our notions of distinct result of a program are all preserved by the semantics. The last intuitively states that if M has a non-erroring behavior, then M' exhibits the same behavior.

4 IDEALIZED DOUBLE CATEGORICAL MODELS OF GRADUALITY

In order to organize our construction of denotational models we first develop sufficient *abstract* categorical semantics of gradually typed languages. We start by modeling the type and term structure of gradual typing and then extend this to type and term precision. Gradually typed languages inherently involve computational effects of errors and non-termination and typically in practice many others such as mutable state and I/O. To model this cleanly categorically, we follow New, Licata and Ahmed's GTT calculus and base our models off of Levy's Call-by-push-value (CBPV) calculus which is a standard model of effectful programming [18]. There are several notions of model of CBPV from the literature with varying requirements of which connectives are present [9, 11, 18], we will

use a variant which models precisely the connectives we require and no more $(1, \times, F, U, \rightarrow)$[1].

(1) A cartesian category $\mathcal{V}$ and a category $\mathcal{E}$.
(2) An action of $\mathcal{V}^{op}$ (with the $\mathcal{V}$ cartesian product as monoidal structure) on $\mathcal{E}$. We write this with an arrow $A \rightarrow B$. This means we have natural isomorphisms $\alpha : A_1 \times A_2 \rightarrow B \cong A_2 \rightarrow (A_1 \rightarrow B)$ and $i : 1 \rightarrow B \cong B$ satisfying pentagon and triangle identities[17].
(3) $F \dashv U$ where $U : \mathcal{E} \rightarrow \mathcal{V}$ such that U "preserves powering" in that every $U(A \rightarrow B)$ is an exponential of UB by A and that $U\alpha$ and Ui are mapped to the canonical isomorphisms for exponentials.

Example 4.1. Given a strong monad T on a bicartesian closed category $\mathcal{V}$, we can extend this to a CBPV model by defining $\mathcal{E}$ to be the category $\mathcal{V}^T$ of algebras of the monad, defining $A \rightarrow B$ as the powering of algebras, F as the free algebra and U as the underlying object functor.

To additionally model the error terms, we add a requirement that there is a natural transformation $\mathfrak{V} : 1 \rightarrow U$. The naturality requirement encodes that strict morphisms (e.g., the denotations of evaluation contexts) preserve errors.

We can then model CBV terms and types in a straightforward adaptation of Levy's interpretation of CBV in CBPV. We interpret types A as objects $A \in \mathcal{V}$ and CBV terms $\Gamma \vdash M : A$ as morphism of any of the equivalent forms $\mathcal{E}(F(\times\Gamma), F(A)) \cong \mathcal{V}(\times\Gamma \vdash UF(A)) \cong \mathcal{E}(F(1), \Gamma \rightarrow F(A))$. The most interesting type translation is the CBV function type: $A \rightharpoonup A' = U(A \rightarrow FA')$. Such a model validates all type-based equational reasoning, i.e., $\beta\eta$ equality, and models the introduction and elimination rules for CBV. Thus a CBPV model is sufficient to interpret the CBV term language. We will require additional structure to interpret the precision and type casts.

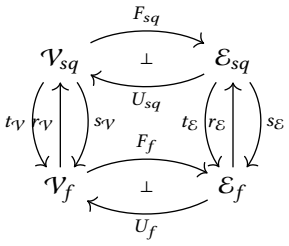
4.1 Double Categorical Semantics of Graduality

New and Licata modeled the graduality and type casts for call-by-name gradual typing using *double categories*, which are defined to be categories internal to the category of categories. That is, a double category C consists of a category C_o of "objects and function morphisms" and a category C_{sq} of "relation morphisms and squares" with functors (reflexive relation) $r : C_o \rightarrow C_{sq}$ and (source and target) $s, t : C_{sq} \rightarrow C_o$ satisfying $sr = tr = \mathrm{id}$ as well as a composition operation $c : C_{sq} \times_{s,t} C_{sq} \rightarrow C_{sq}$ respecting source and target. This models an abstract notion of functions and relations. For notation, we write function morphisms as $f : A \rightarrow B$ and relation morphisms as $c : A \circ\!\!-\!\!\bullet\ B$ where $c \in C_{sq}$ and $s(c) = A$ and $t(c) = B$. Finally a morphism α from c to d with $s(\alpha) = f$ and $s(\beta) = g$ is visualized as

$$\begin{array}{ccc} A & \xrightarrow{c} & B \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle g} \\ A' & \xrightarrow[d]{} & B' \end{array}$$

And is thought of as an abstraction of the notion of relatedness of functions: functions take related inputs to related outputs. The composition operations and functoriality give us a notion of composition of relations as well as functions and vertical and horizontal composition of squares. In this work we will be chiefly interested in *locally thin* double categories, that is, double categories where there is at most one square for any f, c, g, d. In this case we use the notation $f \leq_{c,d} g$ to mean that a square like the above exists.

New, Licata and Ahmed [28] extended the axiomatic syntax to call-by-push-value but did not analyze the structure categorically. We fill in this missing analysis now: a model of the congruence rules of their system can be given by a locally thin "double CBPV model", which we define as a category internal to the category of CBPV models and *strict* homomorphisms of CBPV models[2]. A strict homomorphism of CBPV models from $(\mathcal{V}, \mathcal{E}, \ldots)$ to $(\mathcal{V}', \mathcal{E}', \ldots)$ consists of functors $G_v : \mathcal{V} \rightarrow \mathcal{V}'$ and $G_e : \mathcal{E} \rightarrow \mathcal{E}'$ that strictly preserve all CBPV constructions, see the appendix for a more detailed definition. We call this a strict morphism in contrast to a *lax* morphism, which only preserves CBPV constructions up to transformation. Some of the data of a double CBPV model can be visualized as follows:



Type precision $A \sqsubseteq A'$ is interpreted as a relation morphism $c_A : A \circ\!\!-\!\!\bullet\ A'$ in $\mathcal{V}_{sq}$, and term precision $\Gamma \sqsubseteq \Gamma' \vdash M \sqsubseteq M' : A \sqsubseteq A'$ is interpreted as a square $M \sqsubseteq_{c_\Gamma, UFc_A} M'$. The fact that t, r and the composition are all given by strict CBPV homomorphisms says that all the type constructors lift to precision (monotonicity of type constructors) as well as all term constructors (congruence). Further, r and composition being strict homomorphisms implies that all type constructors strictly preserve the identity relation (identity extension) and composition.

Next, to model type casts, their model further requires that every value relation $c : A \circ\!\!-\!\!\bullet\ A'$ is *left representable* by a function $u_c : A \rightarrow A'$ and every computation relation $d : B \circ\!\!-\!\!\bullet\ B'$ is *right representable* by a function $d_c : B' \rightarrow B$. In a locally thin double category, these are defined as follows:

Definition 4.2. $c : A \circ\!\!-\!\!\bullet\ B$ is left representable by $f : A \rightarrow B$ if $f \sqsubseteq^{c}_{r(B)} \mathrm{id}$ and $\mathrm{id} \sqsubseteq^{r(A)}_{c} f$.

Dually, $c : A \circ\!\!-\!\!\bullet\ B$ is right representable by $g : B \rightarrow A$ if $\mathrm{id} \sqsubseteq^{c}_{r(A)} g$ and $g \sqsubseteq^{r(B)}_{c} \mathrm{id}$.

These rules are sufficient to model the UpL/UpR/DnL/DnR rules for casts. Additionally, since representable morphisms compose and so the compositionality of casts comes for free. However, the retraction property must be added as an additional axiom to the model. To model the error being a least element we add the requirement that $\mathfrak{V} \circ ! \sqsubseteq^{r(A)}_{r(UB)} f$ holds for all $f : \mathcal{V}(A, B)$. Finally, the

[1] It is essential in this case that we do not require a cartesian closed category of values as there is no way to implement casts for an exponential in general.

[2] it may be possible to also define this as a notion of CBPV model internal to some structured 2-category of categories, but the authors are not aware of any such definition of an internal CBPV model

dynamic type can be modeled as an arbitrary value type D with arbitrary relations $\text{Nat} \circ\!\!-\!\!\bullet D$ and $D \rightharpoonup D \circ\!\!-\!\!\bullet D$ and $D \times D \circ\!\!-\!\!\bullet D$ (or whatever basic type cases are required).

Example 4.3. (Adapted from [27]): Define a double CBPV model where $\mathcal{V}$ is the category of predomain preorders: sets with an ω-CPO structure $\leq$ as well as a poset structure $\sqsubseteq$. Functional morphisms are given by $\leq$-continuous and $\sqsubseteq$-monotone functions. Then define $\mathcal{E}$ to be the category of pointed domain preorders which are domain preorders with least elements $\bot$ for $\leq$ and $\mho$ for $\sqsubseteq$ such that $\bot$ is $\sqsubseteq$-maximal, and morphisms are as before but preserve $\bot$ and $\mho$. This can be extended to a CBPV model with forgetful functor $U : \mathcal{E} \to \mathcal{V}$. D can be defined by solving a domain equation.

This can be extended to a double CBPV model by defining a value relation $A \circ\!\!-\!\!\bullet A'$ to be a $\sqsubseteq$-*embedding*: a morphism $e : A \to A'$ that is injective and such that $Fe : FA \to FA'$ has a right adjoint (with respect to $\sqsubseteq$) and a square $f \sqsubseteq^{e}_{e'} f' = f \circ e \sqsubseteq e' \circ f$. Similarly computation relations $B \circ\!\!-\!\!\bullet B'$ are defined to be *projections*: morphisms $p : B' \to B$ that are surjective and Up has a left adjoint, with squares defined similarly. A suitable dynamic type can be constructed by solving a domain equation $D \cong \text{Nat} + U(D \to FD) + (D \times D)$.

4.2 Weakening the Double Category Semantics

While the double categorical semantics can be satisfied with classical domain theoretic models, there are obstructions to developing a semantics based on *guarded* recursion. While guarded type theory makes construction of arbitrary guarded recursive definitions possible, it comes at a significant cost to reasoning: unfolding of recursive definitions is explicit. In a denotational semantics, this means that non-well-founded recursion must perform a kind of *observable step*. This is a difficulty in proving graduality, which is a property that is oblivious to the number of steps that a program takes. Therefore to prove graduality we must impose a kind of weak bisimilarity relation on our programs to reason about step-independent relational properties. However, here we arrive at the fundamental issue with double categorical semantics using representable relations:

(1) When reasoning up to weak bisimilarity, transitive reasoning is not possible, and so horizontal pasting of squares is not a valid reasoning principle.
(2) When reasoning with observable computation steps, certain casts take observable steps, and can no longer be modeled using representable morphisms.

Our solution to this dilemma comes in two parts. Since graduality ignores computational steps, the syntactic theory of graduality must be modeled up to weak bisimilarity, where transitive reasoning is not valid. For this purpose we develop a notion of *extensional* model which weakens from double categories to a reflexive graph categories, dropping the operation of horizontal pasting of squares, but still maintaining a form of representability of relations.

However, transitive reasoning is essential for compositional semantic constructions, and so we need to work with an intermediate notion of *intensional* model which is based on double categories, but where representability has to be weakened to "quasi-representability", a kind of representability *up to* observable steps. To reason up to observable steps without using weak bisimilarity, we develop a notion we call *perturbations*, certain terms that are bisimilar to the identity but can be manipulated explicitly in constructions. We then show that the compositional construction of casts from domain theoretic models can be adapted to this guarded setting by incorporating some explicit manipulation of perturbations. Finally, we show that taking an "extensional collapse" by bisimilarity provides a model of the extensional theory, which can then be used to model the global graduality property.

5 REVISED CATEGORICAL MODELS OF GRADUALITY

Next, we develop our appropriate weakened notions of models of graduality, which we divide into *extensional* models, where ordering is up to weak bisimilarity, and *intensional* models, where ordering relates terms that considers computational steps to be observable. We develop the notion of intensional model of gradual typing in stages and show how to develop one from a base model of effectful functions and relations.

5.1 Extensional Models of Gradual Typing

Since we lack transitivity of ordering when reasoning in guarded type theory, our weakened notion of extensional model is based on reflexive graph categories rather than double categories. This means we lose the reasoning principle of horizontal pasting of squares. We will still require a notion of *composition* of relations, to model the transitivity of type precision. We note that without horizontal pasting of squares, the notion of left/right representability of squares is not sufficient to interpret the cast rules of gradual typing. Instead we generalize the notion of representability to match the syntactic rules in Section 3.

Let $c : A \circ\!\!-\!\!\bullet A'$ and $f : A \to A'$ in a reflexive graph category with composition of relations. We say that c is *universally left-representable by* f if for any $c_l : A_l \circ\!\!-\!\!\bullet A$ and $c_r : A' \circ\!\!-\!\!\bullet A_r$ we have $f \sqsubseteq^{cc_r}_{c_r}$ id and id $\sqsubseteq^{c_l}_{c_l c}$ f. Dually, let $d : B \circ\!\!-\!\!\bullet B'$ and $g : B' \to B$. We say that d is *universally right-representable by* g if for any $d_l : B_l \circ\!\!-\!\!\bullet B$ and $d_r : B' \circ\!\!-\!\!\bullet B_r$ we have id $\sqsubseteq^{d_l d}_{d_l}$ ϕ and $\phi \sqsubseteq^{d_r}_{dd_r}$ id. In a reflexive graph category these are stronger than the definitions of left/right representability since we can pick c_l, c_r, d_l, d_r to be the reflexive relations. In a double category these are equivalent, but the equivalence uses horizontal pasting.

Additionally, while in the presence of horizontal pasting compositionality of representable morphisms is automatic, without this principle we must require it explicitly. Finally, we explicitly add in a requirement that the type constructors are functorial in the

In summary, an extensional model consists of:

(1) A locally thin reflexive graph internal to CBPV models.
(2) Composition of value and computation relations that form a category with the reflexive relations as identity. Call these categories $\mathcal{V}_r, \mathcal{E}_r$
(3) Identity-on-objects functors up : $\mathcal{V}_r \to \mathcal{V}_f$ and dn : $\mathcal{E}_r^{op} \to \mathcal{E}_f$ such that every up$c$ universally left-represents c and every dn$d$ universally represents d.

(4) The CBPV connectives $U, F, \times, \to$ are all *covariant* functorial on relations up to equivalence: $U(dd') \sqsupseteq\sqsubseteq U(d)U(d')$ etc.[3] where $c \sqsupseteq\sqsubseteq c'$ means $\mathsf{id} \sqsubseteq^{c}_{c'} \mathsf{id}$ and $\mathsf{id} \sqsubseteq^{c'}_{c} \mathsf{id}$.
(5) A natural transformation $\mathfrak{V} : 1 \Rightarrow U$ such that $\mathfrak{V} \circ ! \sqsubseteq^{r(A)}_{r(UB)} f$ for any $f : A \to UB$
(6) Distinguished value type Nat with morphisms $z : \mathcal{V}(1, \mathrm{Nat})$ and $s : \mathcal{V}(\mathrm{Nat}, \mathrm{Nat})$.
(7) Distinguished value types D with distinguished relations $\mathsf{Inj}_{\to} : U(D \to FD) \circ\!\!-\!\!\bullet D$ and $\mathsf{Inj}_{\mathbb{N}} : \mathrm{Nat} \circ\!\!-\!\!\bullet D$ and $\mathsf{Inj}_{\times} : D \times D \circ\!\!-\!\!\bullet D$ each satisfying the retraction property $\mathsf{dn}\, \mathsf{Inj}_{\to}()\, F(\mathsf{up}\, \mathsf{Inj}_{\to}())\, \sqsupseteq\sqsubseteq \mathsf{id}$.

This mathematical structure is sufficient to interpret the theory of gradual typing in Section 3.

Definition 5.1. Given any extensional gradual typing model, we interpret

(1) Each type A as a value type, interpreting the base types and $\times$ as their semantic analogues and $[\![A \rightharpoonup A']\!]$ as $U([\![A]\!] \to F([\![A']\!]))$.
(2) Type precision derivations $c : A \sqsubseteq A'$ are interpreted as relation morphisms $[\![c]\!] : [\![A]\!] \circ\!\!-\!\!\bullet [\![A']\!]$ in the obvious way. Every equivalence axiom $c \equiv c'$ implies that $[\![c]\!] \sqsupseteq\sqsubseteq [\![c']\!]$.
(3) Every term $\Gamma \vdash M : A$ is interpreted as a morphism $[\![M]\!] : \mathcal{V}_f(\times[\![\Gamma]\!], UF[\![A]\!])$. Upcasts are interpreted as $UF(u_{[\![c]\!]})$ and downcasts as $Ud_{F[\![c]\!]}$.
(4) If $\Delta \vdash M \sqsubseteq M' : c$ then $[\![M]\!] \sqsubseteq^{\times[\![\Delta]\!]}_{[\![c]\!]} [\![M']\!]$ holds.

5.2 Intensional Models

An intensional model of gradual typing is defined similarly to an extensional model, with a few key differences that will be discussed below. The starting point is similar to that of the extensional model. This time, however, since we are working intensionally, the semantic denotation of term precision *is* transitive, so we *do* have a horizontal composition operation on squares. Compare this to the extensional case, where we could only compose *relations* horizontally, not squares. We can specify this compactly as a category internal to the category of CBPV models and lax morphisms, where we require that the reflexivity, source, and target morphisms are strict. In particular, as in the extensional case, there is a CBPV model of "objects" $\mathcal{M}_f$ and a CBPV model of "arrows" $\mathcal{M}_{sq}$. There are strict CBPV morphisms $r : \mathcal{M}_f \to \mathcal{M}_{sq}$ and $s, t : \mathcal{M}_{sq} \to \mathcal{M}_f$, just as before. But now, we also have a CBPV morphism m from the pullback $\mathcal{M}_{sq} \times_{s=t} \mathcal{M}_{sq}$ to M_{sq}, i.e., "composition of arrows". In particular, this consists of a functor $m_{\mathcal{V}} : \mathcal{V}_{sq} \times_{s_{\mathcal{V}}=t_{\mathcal{V}}} \mathcal{V}_{sq} \to \mathcal{V}_{sq}$ for composition of value relations/squares, and a functor $m_{\mathcal{E}} : \mathcal{E}_{sq} \times_{s_{\mathcal{E}}=t_{\mathcal{E}}} \mathcal{E}_{sq} \to \mathcal{E}_{sq}$ for composition of computation relations/squares. Furthermore, $s \circ m = s \circ \pi_1$ and $t \circ m = t \circ \pi_2$.

As in the extensional model, we also require the existence of a natural transformation $\mathfrak{V} : 1 \Rightarrow U$ such that $\mathfrak{V} \circ ! \sqsubseteq^{r(A)}_{r(UB)} f$ for any $f : A \to UB$, and a distinguished value type Nat with morphisms $z : \mathcal{V}(1, \mathrm{Nat})$ and $s : \mathcal{V}(\mathrm{Nat}, \mathrm{Nat})$.

[3] the reflexive graph structure already requires that these functors preserve identity relations

5.2.1 Bisimilarity. Working intensionally means we need to take into consideration the steps taken by terms. One consequence of this is that we need a way to specify that two morphisms are the same "up to delay", i.e., they differ only in that one may wait more than the other.

In particular, for any pair of objects A and A', in $\mathcal{V}_f$, we require that there is a reflexive, symmetric relation $\approx_{A,A'}$ on the hom-set $\mathcal{V}_f(A, A')$, called the *weak bisimilarity* relation. Similarly for the computation category: there is a reflexive, symmetric relation $\approx_{B,B'}$ defined on each hom-set $\mathcal{E}_f(B, B')$. Additionally, the weak bisimilarity relation should respect composition: if $f \approx_{A,A'} f'$ and $g \approx_{A',A''} g'$, then $g \circ f \approx_{A,A''} g' \circ f'$, and likewise for computations.

Lastly, we require that for any computation object B, the hom-set $\mathcal{V}_f(UB, UB)$ contains a distinguished morphism δ^*_{UB}, such that $\delta^*_{UB} \approx_{UB,UB} \mathsf{id}_{UB}$. Moreover, we require that these morphisms are related in that for any $d : B \circ\!\!-\!\!\bullet B'$, we have a square $\delta^*_{UB} \sqsubseteq^{Ud}_{Ud} \delta^*_{UB'}$. We also require that these morphisms commute with computation morphisms, in the sense that for any $\phi \in \mathcal{E}_f(B, B')$ we have $U\phi \circ \delta^*_{UB_1} = \delta_{UB_2} \circ U\phi$. Given the existence of the morphisms δ^*_{UB}, we can define a computation morphism $\delta^*_{FA} \in \mathcal{E}_f(FA, FA)$ for all A by composing the unit $\eta_A \in \mathcal{V}_f(A, UFA)$ with the morphism $\delta^*_{UFA} \in \mathcal{V}_f(UFA, UFA)$, and then by the adjunction we get a computation morphism $\delta^{\in}_{FA} \mathcal{E}_f(FA, FA)$. Moreover, we get a square $\delta^*_{FA} \sqsubseteq^{Fc}_{Fc} \delta^*_{FA'}$ for all $c : A \circ\!\!-\!\!\bullet A$.

Definition 5.2. A *step-1 intensional model* consists of all the data of a step-0 intensional model along with:

- The existence of a reflexive, symmetric relation $\approx$ on the hom-sets $\mathcal{V}_f(A, A')$ and $\mathcal{E}_f(B, B')$ such that $\approx$ preserves composition.
- The existence of a distinguished value morphism $\delta^*_{UB} \approx \mathsf{id}_{UB}$ for each B.
- A square $\delta^*_{UB} \sqsubseteq^{Ud}_{Ud} \delta^*_{UB'}$ for all $d : B \circ\!\!-\!\!\bullet B'$.
- The commutativity condition $U\phi \circ \delta^*_{UB_1} = \delta_{UB_2} \circ U\phi$ for any $\phi \in \mathcal{E}_f(B, B')$.

5.2.2 Perturbations. A second consequence of working intensionally is that the squares in the representable properties must now involve a notion of "delay" or "perturbation" in order to keep the function morphisms on each side in lock-step. Intuitively, the perturbations have no effect other than to cause the function to which they are applied to "wait" in a specific manner. We formalize this notion by requiring that for each object A in $\mathcal{V}_f$, there is a monoid of *value perturbations* P_A and a homomorphism of monoids $\mathrm{ptb}_A : P_A \to \{f \in \mathcal{V}_f(A, A) \mid f \approx \mathsf{id}\}$. Similarly, for each $B : \mathcal{E}_f$ there is a monoid $P^{\mathcal{E}}_B$ of *computation perturbations* and a homomorphism of monoids $\mathrm{ptb}_B : P_B \to \{g \in \mathcal{E}_f(B, B) \mid g \approx \mathsf{id}\}$.

Note that we require that all perturbations be weakly bisimilar to the identity morphism, capturing the notion that they have no effect other than to delay.

We will slightly abuse notation and refer to an endomorphism $f \in \mathcal{V}_f(A, A)$ as being "in" the monoid of perturbations, by which we actually mean there is an element $p \in P_A$ that is mapped to f under the homomorphism.

We require that $\delta^*_{UB} \in P_{UB}$ for all B, where δ^*_{UB} is the distinguished morphism that is required to be present in every hom-set $\mathcal{V}_f(UB, UB)$ per the definition of a step-1 model.

The perturbations must be preserved by $\times \overset{\text{k}}{\times}, \to, \overset{\text{k}}{\to}, U, \overset{\text{k}}{U}, F$, and $\overset{\text{k}}{F}$. Perturbations must also satisfy a property that we call the "push-pull" property, which is formulated as follows. Let $c : A \circ\!\!-\!\!\bullet A'$. Given a perturbation $\delta \in P_A$, there is a corresponding perturbation $\text{push}_c(\delta) \in P_{A'}$. Likewise, given $\delta' \in P_{A'}$ there is a perturbation $\text{pull}_c(\delta') \in P_A$. Moreover, push-pull states that there are squares $\delta \sqsubseteq^c_c \text{push}_c(\delta)$ and $\text{pull}_c(\delta') \sqsubseteq^c_c \delta'$.

The analogous property should also hold for computation relations and perturbations. The requirements are summarized below:

Definition 5.3. A *step-2* model of intensional gradual typing consists of all the data of a step-1 model plus:

(1) For each value type A, there is a monoid P_A and a homomorphism of monoids $\text{ptb}_A : P_A \to \{f \in \mathcal{V}_f(A, A) \mid f \approx \text{id}\}$.
(2) For each computation type B, there is a monoid $P^{\mathcal{E}}_B$ and a homomorphism of monoids $\text{ptb}_B : P_B \to \{g \in \mathcal{E}_f(B, B) \mid g \approx \text{id}\}$.
(3) For all B, the distinguished endomorphism δ^*_{UB} is in P_{UB}.
(4) The functors $\times, \overset{\text{k}}{\times}, \to, \overset{\text{k}}{\to}, U, \overset{\text{k}}{U}, F, \overset{\text{k}}{F}$ preserve perturbations.
(5) The push-pull property holds for all $c : A \circ\!\!-\!\!\bullet A'$ and all $d : B \circ\!\!-\!\!\bullet B'$.

5.2.3 Behavior of Casts. As is the case in the extensional model, there is a relationship between vertical (i.e., function) morphisms and horizontal (i.e., relation) morphisms, but as mentioned above, now there are perturbations involved in order to keep both sides "in lock-step". We begin with a step-2 intensional model as defined in the previous section, and provide the additional conditions that axiomatize the behavior of casts. The precise definitions are as follows.

First, let $\mathcal{M}$ be any double category with a notion of perturbations, i.e., for any object X in $\mathcal{M}$ there is a monoid P_X with a monoid homomorphism into the endomorphisms on X.

Definition 5.4. Let R be a horizontal morphism in $\mathcal{M}$ between objects X and Y. We say that R is quasi-left-representable by a vertical morphism f in $\mathcal{M}(X, Y)$ if there are perturbations $\delta^{l,e}_R \in P_X$ and $\delta^{r,e}_R \in P_Y$ such that there is a square UpL : $f \sqsubseteq^R_{r(Y)} \delta^{r,e}_R$ and a square UpR : $\delta^{l,e}_R \sqsubseteq^{r(X)}_R$

Definition 5.5. Let R be a horizontal morphism between X and Y. We say that R is *quasi-right-representable by* $f \in \mathcal{M}(Y, X)$ if there exist perturbations $\delta^{l,p}_R \in P_X$ and $\delta^{r,p}_R \in P_Y$ such that we have a square DnR : $\delta^{l,p}_R \sqsubseteq^R_{r(X)} f$ and a square DnL : $f \sqsubseteq^{r(Y)}_R \delta^{r,p}_R$.

With these definitions, we return to the more specific setting of a step-2 intensional model $\mathcal{M}$ and specify the new requirements for relations. We require that there are functors up : $\mathcal{V}_e \to \mathcal{V}_f$ and dn : $\mathcal{E}^{op}_e \to \mathcal{E}_f$ Every value edge $c : A \circ\!\!-\!\!\bullet A'$ must be quasi-left-representable by $\text{up}(c)$, and every computation edge $d : B \circ\!\!-\!\!\bullet B'$ is quasi-right-representable by $\text{dn}(d)$.

Besides the perturbations, one other difference between the extensional and intensional versions of the representability axioms is that in the extensional setting, the rules build in the notion of composition, whereas their intensional counterparts do not. In the extensional setting, we do not have horizontal composition of squares, which is required to derive the versions of the rules that build in composition from the versions that do not. In the intensional setting, we do have horizontal composition of squares, so we can take the simpler versions as primitive and derive the ones involving composition.

Lastly, we require that the model satisfy a weak version of functoriality for the CBPV connectives $U, F, \times, \to$. First, we will need a definition:

Definition 5.6 (quasi-order-equivalence). Let $c, c' : A \circ\!\!-\!\!\bullet A'$. We say that c and c' are *quasi-order-equivalent*, written $c \approx c'$, if there exist perturbations $\delta^l_1, \delta^l_2 \in P^{\mathcal{V}}_A$ and $\delta^r_1, \delta^r_2 \in P^{\mathcal{V}}_{A'}$ such that there is a square $\delta^l_1 \sqsubseteq^c_{c'} \delta^r_1$ and a square $\delta^l_2 \sqsubseteq^{c'}_c \delta^r_2$. We make the analogous definition for computation relations $d, d' : B \circ\!\!-\!\!\bullet B'$.

We require that the CBPV connectives $U, F, \times, \to$ are *quasi-functorial* on relations, which we specify as follows:

- $U(d \bullet d') \approx U(d)U(d')$
- $F(c \bullet c'') \approx F(c)F(c')$
- $(cc') \to (dd') \approx (c \to d)(c' \to d')$
- $(c_1 c'_1) \times (c_2 c'_2) \approx (c_1 \times c_2)(c'_1 \times c'_2)$

We summarize the requirements of a step-3 model below:

Definition 5.7. A *step-3 intensional model* consists of all the data of a step-2 intensional model, such that additionally:

(1) There are functors up : $\mathcal{V}_e \to \mathcal{V}_f$ and dn : $\mathcal{E}^{op}_e \to \mathcal{E}_f$
(2) Every value edge $c : A \circ\!\!-\!\!\bullet A'$ is quasi-left-representable by $\text{up}(c)$ and every computation edge $d : B \circ\!\!-\!\!\bullet B'$ is quasi-right-representable by $\text{dn}(d)$.
(3) The CBPV connectives $U, F, \times, \to$ are quasi-functorial on relations.

5.2.4 The Dynamic Type. Now we can discuss what it means for an intensional model to model the dynamic type.

Definition 5.8. A step-4 intensional model is a step-3 intensional model $\mathcal{M}$ such that:

(1) There is a distinguished value object $D \in \text{Ob}(\mathcal{V}_f)$.
(2) There are distinguished value relations $\text{Inj}_\to : U(D \to FD) \circ\!\!-\!\!\bullet D$ and $\text{Inj}_{\mathbb{N}} : \text{Nat} \circ\!\!-\!\!\bullet D$ and $\text{Inj}_\times : D \times D \circ\!\!-\!\!\bullet D$ each satisfying the retraction property up to bisimilarity.

5.3 Constructing an Extensional Model

In the previous section, we have given the definition of an intensional model of gradual typing as a series of steps with each definition building on the previous one. Here, we discuss how to construct an extensional model from an intensional model with dyn. We do so in several phases, beginning with a step-1 intensional model with dyn and ending with an extensional model. Moreover, this construction is *modular*, in that each phase of the construction does not depend on the details of the previous ones.

5.3.1 Adding Perturbations. Suppose we have a step-1 intensional model $\mathcal{M}$. Recall that a step-1 intensional model consists of a step-0 model (i.e., a category internal to the category of CBPV models), along with the necessary categories and functors for bisimilarity as discussed in Section 5.2.1. Further, recall that a step-2 model has everything a step-1 model has, with the addition of perturbation monoids $P^{\mathcal{V}}_A$ for all A and $P^{\mathcal{E}}_B$ for all B. Moreover, the push-pull

property must hold for all value relations c and all computation relations d.

We claim that from a step-1 model, we can construct a step-2 model. The value objects of the model are defined to be triples (A, P_A, ptb_A) where A is a value object in $\mathcal{M}$, P_A is a monoid and ptb_A is a homomorphism of monoids from P_A to the endomorphisms on A that are bisimilar to the identity. Likewise, computation objects are triples (B, P_B, ptb_B). The morphisms are the same as the morphisms of $\mathcal{M}$. A value relation between (A, P_A, ptb_A) and $(A', P_{A'}, \mathrm{ptb}_{A'})$ is given by a pair of a relation in c and a *push-pull structure* Π_c specifying that that c satisfies the push-pull property. Computation relations are defined analogously. The squares are the same as those of $\mathcal{M}$. We define the action of the functor F on objects as $F(A, P_A, \mathrm{ptb}_A) = (FA, \mathbb{N} \times P_A, \mathrm{ptb}_{FA})$ where $\mathrm{ptb}_{FA}(n, a) = (\delta^*_{FA})^n \circ F(\mathrm{ptb}_A(a))$ (i.e., we use the distinguished delay morphism δ^*_{FA}). The action of U on objects is defined similarly, where the perturbations are defined to be $\mathbb{N} \times P_B$.

For the full details of the construction, see Lemma C.1 in the Appendix.

5.3.2 Adding Quasi-Representability. Now suppose we have a step-2 intensional model $\mathcal{M}$. We claim that we can construct a step-3 intensional model $\mathcal{M}'$. In the construction, the objects and morphisms are the same as those of $\mathcal{M}$, and a value relation between A and A' consists of a triple $(c, \rho^L_c, \rho^R_{Fc})$ where $c : A \circ\!\!-\!\!\bullet\, A'$ is a relation in $\mathcal{M}$, ρ^L_c is a quasi-*left*-representation for c (i.e., an embedding e_c, perturbations $\delta^{r,e}_c \in P_{A'}$ and $\delta^{l,e}_c \in P_A$, and the two relevant squares), and similarly ρ^R_{Fc} is a quasi-*right*-representation for Fc. Computation relations are triples $(d, \rho^R_d, \rho^L_{Ud})$ where $d : B \circ\!\!-\!\!\bullet\, B'$, ρ^R_d is a quasi-right-representation for d and $\rho^L Ud$ is a quasi-left-representation for Ud. The value and computation squares are the same as those of $\mathcal{M}$. We then define composition of relations and the action of the functors $F, U, \times$, and $\to$.

For the full details of the construction, see Lemma C.7 in the Appendix.

5.3.3 Constructing an Extensional Model. Finally, suppose $\mathcal{M}$ is a step-4 intensional model (i.e., a step-3 model with an interpretation of the dynamic type). We now describe how to build an extensional model.

The idea is to define an extensional model whose squares are the "bisimilarity-closure" of the squares of the provided intensional model $\mathcal{M}$.

The categories $\mathcal{V}_f, \mathcal{E}_f$ are the same as those of $\mathcal{M}$. Additionally, the objects of $\mathcal{V}_{sq}$ and $\mathcal{E}_{sq}$, i.e., the value and computation relations, are the same. The difference arises in the *morphisms* of $\mathcal{V}_{sq}$ and $\mathcal{E}_{sq}$, i.e., the commuting squares. In particular, a morphism $\alpha_e \in \mathcal{V}'_{sq}(c_i, c_o)$ with source f and target g is given by:

- a morphism $f' \in \mathcal{V}_f(A_i, A_o)$ with $f \approx f'$.
- a morphism $g' \in \mathcal{V}_f(A'_i, A'_o)$ with $g \approx g'$.
- a square $\alpha_i \in \mathcal{V}_{sq}(c_i, c_o)$ with source f' and target g'.

Using our existing notation, we say that $f \lesssim^{c_i}_{c_o} g$ if there exist f' and g' such that

$$f \approx_{A_i, A_o} f' \sqsubseteq^{c_i}_{c_o} g' \approx_{A'_i, A'_o} g.$$

We make the analogous construction for the computation squares.

The proof that this indeed defines an extensional model is given in the Appendix (see Section C.3).

6 CONSTRUCTING A CONCRETE MODEL

In this section, we build a concrete extensional model of gradual typing. We begin by defining a step-1 intensional model, and then apply the abstract constructions outlined in the previous section to obtain an extensional model. We begin with some definitions:

Definition 6.1. A **predomain** A consists of a set A along with two relations:

- A partial order $\leq_A$.
- A reflexive, symmetric "bisimilarity" relation $\approx_A$.

Given a predomain A, we can form the predomain $\triangleright\, A$. The underlying set is $\triangleright\, |A|$ and the relation is defined in the obvious way, i.e., $\tilde{x} \leq_{\triangleright A} \tilde{x'}$ iff $\triangleright_t\ (\tilde{x}_t \leq_A \tilde{x'}_t)$. Likewise for bisimilarity. We also give a predomain structure to the natural numbers $\mathbb{N}$, where both the ordering and the bisimiarity relation are equality. Morphisms of predomains are functions between the underlying sets that preserve the ordering and the bisimilarity relation. More formally:

Definition 6.2. Let A and A' be predomains. A morphism $f : A \to A'$ is a function between the underlying sets such that for all x, x', if $x \leq_A x'$, then $f(x) \leq f(x')$, and if $x \approx_A x'$, then $f(x) \approx_{A'} f(x')$.

Definition 6.3. An **error domain** B consists of a predomain B along with the following data:

- A distinguished "error" element $\mathfrak{V}_B \in B$
- A morphism of predomains $\theta_B : \triangleright\, B \to B$

For an error domain B, we define the predomain morphism $\delta_B := \theta_B \circ \mathrm{next}$. Morphisms of error domains are morphisms of the underlying predomains that preserve the algebraic structure. More formally:

Definition 6.4. Let B and B' be error domains. A morphism $\phi : B \multimap B'$ is a morphism between the underlying predomains such that:

(1) $\phi(\mathfrak{V}_B) = \mathfrak{V}_{B'}$
(2) $\phi(\theta_B(\tilde{x})) = \theta_{B'}(\lambda t.\phi(\tilde{x}_t))$

We define a (monotone) relation on predomains A and A' to be a relation on the underlying sets that is downward-closed under $\leq_A$ and upward-closed under $\leq_{A'}$. More formally:

Definition 6.5. Let A and A' be predomains. A *predomain relation* between A and A' is a relation R between the underlying sets such that:

(1) (Downward closure): For all $x_1, x_2 \in A$ and $y \in A'$, if $x_1 \leq_A x_2$ and $x_2\ R\ y$, then $x_1\ R\ y$.
(2) (Upward closure): For all $x \in A$ and $y_1, y_2 \in A'$, if $x\ R\ y_1$ and $y_1 \leq_{A'} y_2$, then $x\ R\ y_2$.

Composition of relations on predomains is the usual relational composition. Similarly, we define a (monotone) relation on error domains to be a relation on the underlying predomains that respects error and preserves θ.

Definition 6.6. Let B and B' be error domains. An *error domain relation* between B and B' is a relation R between the underlying predomains such that

(1) (Respects error): For all $y \in B'$, we have $\mathfrak{U}_B \; R \; y$.
(2) (Preserves θ): For all $\tilde{x}$ in $\triangleright \; B$ and $\tilde{y} \in \triangleright \; B'$, if $\triangleright_t \; (\tilde{x}_t \; R \; \tilde{y}_t)$ then $\theta_B(\tilde{x}) \; R \; \theta_{B'}(\tilde{y})$.

We define composition of error domain relations R on B_1 and B_2 and S on B_2 and B_3 to be the least relation containing R and S that respects error and preserves θ. Specifically, it is defined inductively by the following rules:

$$\frac{b_1 \; R \; b_2 \qquad b_2 \; S \; b_3}{b_1 \; R \odot S \; b_3} \textsc{Comp} \qquad \frac{}{\mathfrak{U}_{B_1} \; R \odot S \; b_3} \textsc{PresErr}$$

$$\frac{\triangleright_t \; (\tilde{b_1} \; R \odot S \; \tilde{b_3})}{\theta_{B_1}(\tilde{b_1}) \; R \odot S \; \theta_{B_3}(\tilde{b_3})} \textsc{PresTheta}$$

We now describe the "commuting squares". Suppose we are given predomains A_i, A_o, A'_i, and A'_o, relations $R_i : A_i \circ\!\!-\!\!\bullet \; A'_i$ and $R_o : A_o \circ\!\!-\!\!\bullet \; A'_o$, and morphisms $f : A_i \to A_o, f' : A'_i \to A'_o$ Given a square with these morphisms and relations, we say that the square commutes, written $f \leq f'$, if for all $x \in A_i$ and $x' \in A'_i$ with $x \; R_i \; x'$, we have $f(x) \; R_o \; f'(x')$. We make the analogous definition for error domains.

6.1 Guarded Lift Monad

The guarded error-lift monad $L_{\mathfrak{U}}$ takes a predomain A to the error domain $L_{\mathfrak{U}}A$. It is defined [4] as follows :

$$L_{\mathfrak{U}}A := \mid \; \eta \colon A \to L_{\mathfrak{U}}A \; \mid \; \mathfrak{U} \colon L_{\mathfrak{U}}A \; \mid \; \theta \colon \; \triangleright \; (L_{\mathfrak{U}}A) \to L_{\mathfrak{U}}A$$

This captures the intuition that a program may either return a value, fail at run-time, or take one or more observable steps of computation. Previous work has studied such a similar construct, called the guarded lift monad [20]; our version here our version augments it with the notion of error. Since we claimed that $L_{\mathfrak{U}}A$ is a monad, we need to define the monadic operations and show that they respect the monadic laws. The return is just η, and the monadic extend is defined via guarded recursion by cases on the input. Verifying that the monadic laws hold uses Löb-induction and is straightforward.

There is a functor U from error domains to predomains that on objects simply returns the underlying predomain, and on morphisms returns the underlying morphism of predomains. It is easily verified that $L_{\mathfrak{U}}A$ is the free error- and later-algebra on the predomain A, so we have that $L_{\mathfrak{U}}$ is left-adjoint to U.

The partial order $\leq_{L_{\mathfrak{U}}A}$ is the lock-step error ordering defined by guarded recursion as follows:

- $\eta \; x \leq_{L_{\mathfrak{U}}A} \eta \; y$ if $x \leq_A y$.
- $\mathfrak{U} \leq_{L_{\mathfrak{U}}A} l$ for all l
- $\theta \; \tilde{r} \leq_{L_{\mathfrak{U}}A} \theta \; \tilde{r'}$ if $\triangleright_t \; (\tilde{r}_t \leq_{L_{\mathfrak{U}}A} \tilde{r'}_t)$

The idea is that two computations l and l' are related if they are in lock-step with regard to their intensional behavior, up to l erroring. Given a relation $R : A \circ\!\!-\!\!\bullet \; A'$, we define in an analogous manner a heterogeneous version of the lock-step error ordering between $L_{\mathfrak{U}}R : L_{\mathfrak{U}}A \circ\!\!-\!\!\bullet \; L_{\mathfrak{U}}A'$.

For a predomain A, we define a relation on $L_{\mathfrak{U}}A$, called "weak bisimilarity", written $l \approx l'$. Intuitively, we say $l \approx l'$ if they are equivalent "up to delay". The weak bisimilarity relation is defined by guarded recursion as follows:

$\mathfrak{U} \approx \mathfrak{U}$

$\eta \; x \approx \eta \; y$ if $x \approx_A y$

$\theta \; \tilde{x} \approx \theta \; \tilde{y}$ if $\triangleright_t \; (\tilde{x}_t \approx \tilde{y}_t)$

$\theta \; \tilde{x} \approx \mathfrak{U}$ if $\theta \; \tilde{x} = \delta^n(\mathfrak{U})$ for some n

$\theta \; \tilde{x} \approx \eta \; y$ if $(\theta \; \tilde{x} = \delta^n(\eta \; x))$ for some n and $x \in A$ such that $x \approx_A y$

$\mathfrak{U} \approx \theta \; \tilde{y}$ if $\theta \; \tilde{y} = \delta^n(\mathfrak{U})$ for some n

$\eta \; x \approx \theta \; \tilde{y}$ if $(\theta \; \tilde{y} = \delta^n(\eta \; y))$ for some n and $y \in A$ such that $x \approx_A y$

When both sides are η, then we ensure that the underlying values are related by the bisimilarity relation on A. When one side is a θ and the other is ηx (i.e., one side steps), we stipulate that the θ-term runs to ηy where x is bisimilar to y. Similarly when one side is θ and the other $\mathfrak{U}$. If both sides step, then we allow one time step to pass and compare the resulting terms. In this way, the definition captures the intuition of terms being equivalent up to delays.

Given predomains A and A', we can form the predomain of predomain morphisms from A to A', denoted $A \Rightarrow A'$.

- The ordering is defined by $f \leq_{A \Rightarrow A'} f'$ iff for all $x \in A$, we have $f(x) \leq_{A'} f'(x)$.
- The bisimilarity relation is defined by $f \approx_{A \Rightarrow A'} f'$ iff for all $x, x' \in A$ with $x \approx_A x'$, we have $f(x) \approx_{A'} f'(x')$.

Given $f : A'_1 \to A_1$ and $g : A_2 \to A'_2$ we define the predomain morphism $f \Rightarrow g : (A_1 \Rightarrow A_2) \to (A'_1 \Rightarrow A'_2)$ by $\lambda h. \lambda x'. g(h(f(x')))$.

We note that $A \Rightarrow UB$ carries a natural error domain structure (in the below, the lambda is a meta-theoretic notation):

- The error is given by $\lambda x. \mathfrak{U}_B$
- The θ operation is defined by

$$\theta_{A \Rightarrow UB}(\tilde{f}) = \lambda x. \theta_B(\lambda t. \tilde{f_t}(x)).$$

Given a predomain A and error domain B, we define $A \to B$ to be the error domain such that $U(A \to B) = A \Rightarrow UB$, and whose error and θ operations are as defined above. We can define the functorial action of $\to$ on morphisms $f \to \phi$ in the obvious way. It is easily verified that $A \to B$ is an exponential of UB by A in the category of predomains and their morphisms.

Lastly, given a relation of predomains R between A and A', and a relation of error domains S between B and B', we define the relation $R \to S$ between $A \to B$ and $A' \to B'$ in the obvious way, i.e., $f \in A \to B$ is related to $g \in A' \to B'$ iff for all $x \in A$ and $x' \in A'$ with $x \; R \; x'$, we have $f(x) \; S \; g(x')$. One can verify that this relation is indeed a relation of error domains in that it respects error and preserves θ.

With all of the above data, we can form a step-1 intensional model of gradual typing (See Definition 5.2).

[4] Formally, the lift monad $L_{\mathfrak{U}}A$ is defined as the solution to the guarded recursive type equation $L_{\mathfrak{U}}A \cong A + 1 + \triangleright \; L_{\mathfrak{U}}A$.

6.2 The Dynamic Type

The predomain representing the dynamic type will be defined using guarded recursion as the solution to the equation [5]

$$D \cong \mathbb{N} + (D \times D) + \triangleright U(D \to FD).$$

For the sake of clarity, we name the "constructors" nat, times, and fun, respectively. We define $e_{\mathbb{N}} : \mathbb{N} \to D$ to be the injection into the first component of the sum, and $e_\times : D \times D \to D$ to be the injection into the second component of the sum, and $e_\to : U(D \to FD)$ to be the morphism next followed by the injection into the third component of the sum.

Explicitly, the ordering on D is given by:

$$\begin{aligned} \mathsf{nat}(n) \leq \mathsf{nat}(n') &\iff n = n' \\ \mathsf{times}(d_1, d_2) \leq \mathsf{times}(d_1', d_2') &\iff d_1 \leq d_2 \text{ and } d_1' \leq d_2' \\ \mathsf{fun}(\tilde{f}) \leq \mathsf{fun}(\tilde{f}') &\iff \triangleright_t\ (\tilde{f}_t \leq \tilde{f}'_t) \end{aligned}$$

We define a relation $\mathrm{Inj}_{\mathbb{N}} : \mathbb{N} \circ\!\!-\!\!\bullet\ D$ by $(n, d) \in \mathrm{Inj}_{\mathbb{N}}$ iff $e_{\mathbb{N}} \leq_D d$. We similarly define $\mathrm{Inj}_\times : D \times D \circ\!\!-\!\!\bullet\ D$ by $((d_1, d_2), d) \in \mathrm{Inj}_\times$ iff $e_\times(d_1, d_2) \leq_D d$, and we define $\mathrm{inj}_\to : U(D \to FD) \circ\!\!-\!\!\bullet\ D$ by $(f, d) \in \mathrm{Inj}_\to$ iff $e_\to(f) \leq_D d$.

Now we define the perturbations for D. Recall from our construction of a model with perturbations (Section 5.3.1) that for each value type A we associate a monoid P_A of perturbations and a homomorphism into the monoid of endomorphisms bisimilar to the identity, and likewise for computation types. We define the perturbations for D via least-fixpoint in the category of monoids as $P_D \cong (P_{D\times D}) \times P_{U(D\to FD)}$. Unfolding the definitions, this is the same as $P_D \cong (P_D \times P_D) \times (\mathbb{N} \times P_D^{op} \times \mathbb{N} \times P_D)$. We now explain how to interpret these perturbations as endomorphisms. We define $\mathrm{ptb}_D : P_D \to \{f : D \to D \mid f \approx \mathsf{id}\}$ below,

$$\begin{aligned} \mathrm{ptb}_D(p_{\mathsf{times}}, p_{\mathsf{fun}}) = \lambda d.\mathsf{case}\ d\ \mathsf{of} \\ &\mid \mathsf{nat}(m) \mapsto \mathsf{nat}(m) \\ &\mid \mathsf{times}(d_1, d_2) \mapsto \mathsf{times}(\mathrm{ptb}_{D\times D}(p_{\mathsf{times}})(d_1, d_2)) \\ &\mid \mathsf{fun}(\tilde{f}) \mapsto \mathsf{fun}(\lambda t.\mathrm{ptb}_{U(D\to FD)}()(\tilde{f}_t)) \end{aligned}$$

One can verify that this defines a homomorphism from $P_D \to \{f : D \to D : f \approx \mathsf{id}\}$. We claim that the three relations $\mathrm{Inj}_{\mathbb{N}}$, $\mathrm{Inj}_\times$, and $\mathrm{Inj}_\to$ satisfy the push-pull property. As an illustrative case, we establish the push-pull property for the relation $\mathrm{Inj}_\to$. We define $\mathrm{pull}_{\mathrm{Inj}_\to} : P_D \to P_{U(D\to FD)}$ $\mathrm{bypull}_{\mathrm{Inj}_\to}(p_{\mathsf{times}}, p_{\mathsf{fun}}) = p_{\mathsf{fun}}$, i.e., we simply forget the other perturbation. We define $\mathrm{push}_{\mathrm{Inj}_\to} : P_{U(D\to FD)} \to P_D$ by $\mathrm{pushInj}_\to(p_{\mathsf{fun}}) = (\mathsf{id}, p_{\mathsf{fun}})$. Showing that the relevant squares commute is straightforward.

We next claim that the relations $\mathrm{Inj}_{\mathbb{N}}$, $\mathrm{Inj}_\times$, and $\mathrm{Inj}_\to$ are quasi-left-representable, and that their lifts are quasi-right-representable. Indeed, since the relations are functional, it is easy to see that they are quasi-left-representable where the perturbations are taken to be the identity. For quasi-right-representability, the most interesting case is $L_{\mathfrak{V}}(\mathrm{Inj}_\to)$. Defining the projection $p_{\mathrm{Inj}_\to} : FD \to FU(D \to FD)$ is equivalent to defining $p' : D \to UFU(D \to FD)$. We define

$$\begin{aligned} p' = \lambda d.\mathsf{case}\ d\ \mathsf{of} &\mid \mathsf{nat}(m) \mapsto \mathfrak{V} \\ &\mid \mathsf{times}(d_1, d_2) \mapsto \mathfrak{V} \\ &\mid \mathsf{fun}(\tilde{f}) \mapsto \theta(\lambda t.\eta(\tilde{f}_t)). \end{aligned}$$

We define $\delta_D^{l,p} = \delta_D^{r,p} = \delta_{FD}$. Then it is easy to show using the definition of $\mathrm{Inj}_\to$ that the squares for DnL and DnR commute. It is also straightforward to establish the retraction property for each of these three relations. In the case of $\mathrm{Inj}_\to$, we have that the property holds up to a delay.

Now that we have defined an intensional model with an interpretation for the dynamic type, we can apply the abstract constructions introduced in Section 5.3. Doing so, we obtain an extensional model of gradual typing, where the squares are given by the "bisimilarity closure" of the intensional error ordering.

6.3 Adequacy

In this section, we prove an adequacy result for the concrete extensional model of GTT we obtained in the previous section. applying the abstract constructions introduced in Section 5.3 to the concrete model built in the previous section.

First we establish some notation. Fix a morphism $f : 1 \to L_{\mathfrak{V}}\mathbb{N} \cong L_{\mathfrak{V}}\mathbb{N}$. We write $f \downarrow n$ to mean that there exists m such that $f = \delta^m(\eta n)$ and $f \downarrow \mathfrak{V}$ to mean that there exists m such that $f = \delta^m(\mathfrak{V})$.

Recall that $\lesssim$ denotes the relation on value morphisms defined as the bisimilarity-closure of the intensional error-ordering on morphisms. That is, we have $f \lesssim g$ iff there exists f' and g' with

$$f \approx_{L_{\mathfrak{V}}\mathbb{N}} f' \leq_{L_{\mathfrak{V}}\mathbb{N}} g' \approx_{L_{\mathfrak{V}}\mathbb{N}} g.$$

Here $\leq_{L_{\mathfrak{V}}\mathbb{N}}$ is the lock-step error ordering, and $\approx_{L_{\mathfrak{V}}\mathbb{N}}$ is weak bisimilarity. First observe that in this ordering, the semantics of error is not equivalent to the semantics of the diverging term. The main result we would like to show is as follows:

Lemma 6.7. *If $f \lesssim g : L_{\mathfrak{V}}\mathbb{N}$, then:*

- *If $f \downarrow n$ then $g \downarrow n$.*
- *If $g \downarrow \mathfrak{V}$ then $f \downarrow \mathfrak{V}$.*
- *If $g \downarrow n$ then $f \downarrow n$ or $f \downarrow \mathfrak{V}$.*

Unfortunately, this is actually not provable! Roughly speaking, the issue is that this is a "global" result, and it is not possible to prove such results inside of the guarded setting. In particular, if we tried to prove the above result in the guarded setting, we would run into a problem where we would have a natural number "stuck" under a $\triangleright$, with no way to get out the underlying number. Thus, to prove our adequacy result, we need to leave the guarded setting and pass back to the more familiar, set-theoretic world with no internal notion of step-indexing. We can do this using a process known as *clock quantification*. Recall that all of the constructions we have made in SGDT take place in the context of a clock k. All of our uses of the later modality and guarded recursion have taken place with respect to this clock. For example, recall the definition of the lift monad by guarded recursion. We can view this definition as being parameterized by a clock k: $L_{\mathfrak{V}}^k : \mathsf{Type} \to \mathsf{Type}$. Then

[5] In this section, we write F instead of $L_{\mathfrak{V}}$ so that the notation follows that of the abstract model section.

for X satisfying a certain technical requirement known as *clock-irrelevance*, [6] we can define the "global lift" monad as $L^{gl}_{\mathcal{U}}X := \forall k.L^k_{\mathcal{U}}X$.

It can be shown that there is an isomorphism between the global lift monad and the delay monad of Capretta [6]. Recall that, given a type X, the delay monad $\mathrm{Delay}(X)$ is defined as the coinductive type generated by $\mathsf{now} : X \to \mathrm{Delay}(X)$ and $\mathsf{later} : \mathrm{Delay}(X) \to \mathrm{Delay}(X)$.

It can be shown that for a clock-irrelevant type X, $L^{gl}_{\mathcal{U}}X$ is a final coalgebra of the functor $F(Y) = X + 1 + Y$ (For example, this follows from Theorem 4.3 in [4].) [7] Since $\mathrm{Delay}(X + 1)$ is also a final coalgebra of this functor, then we have $L^{gl}_{\mathcal{U}}X \cong \mathrm{Delay}(X + 1)$.

Given a predomain X on a clock-irrelevant type, we can define a "global" version of the lock-step error ordering and the weak bisimilarity relation on elements of the global lift; the former is defined by $x \leq^{gl}_X y := \forall k.x[k] \leq y[k]$, and the latter is defined by $x \approx^{gl}_X y := \forall k.x[k] \approx y[k]$. On the other hand, we can define coinductively a "lock-step error ordering" relation on $\mathrm{Delay}(X + 1)$:

$$\frac{}{\mathsf{now}(\mathsf{inr}\ 1) \leq^{\mathrm{Del}} d} \qquad \frac{x_1 \leq_X x_2}{\mathsf{now}(\mathsf{inl}\ x_1) \leq^{\mathrm{Del}} \mathsf{now}(\mathsf{inl}\ x_2)}$$

$$\frac{d_1 \leq^{\mathrm{Del}} d_2}{\mathsf{later}\ d_1 \leq^{\mathrm{Del}} \mathsf{later}\ d_2}$$

And we similarly define by coinduction a "weak bisimilarity" relation on $\mathrm{Delay}(X + 1)$, which uses a relation $d \Downarrow x_?$ between $\mathrm{Delay}(X + 1)$ and $X + 1$ that is defined as $d \Downarrow x_? := \Sigma_{m \in \mathbb{N}} d = \mathsf{later}^m(\mathsf{now}\ x_?)$. Then weak bisimilarity is defined by the rules

$$\frac{x_? \approx_{X+1} y_?}{\mathsf{now}\ x_? \approx^{\mathrm{Del}} \mathsf{now}\ y_?} \qquad \frac{d_1 \Downarrow x_? \qquad x_? \approx_{X+1} y_?}{\mathsf{later}\ d_1 \approx^{\mathrm{Del}} \mathsf{now}\ y_?}$$

$$\frac{d_2 \Downarrow y_? \qquad x_? \approx_{X+1} y_?}{\mathsf{now}\ x_? \approx^{\mathrm{Del}} \mathsf{later}\ d_2} \qquad \frac{d_1 \approx^{\mathrm{Del}} d_2}{\mathsf{later}\ d_1 \approx^{\mathrm{Del}} \mathsf{later}\ d_2}$$

Note the similarity of these definitions to the corresponding guarded definitions. By adapting the aforementioned theorem to the setting of inductively-defined relations, we can show that both the global lock-step error ordering and the global weak bisimilarity admit coinductive definitions. In particular, modulo the above isomorphism between $L^{gl}_{\mathcal{U}}X$ and $\mathrm{Delay}(X + 1)$, the global version of the lock-step error ordering is equivalent to the lock-step error ordering on $\mathrm{Delay}(X + 1)$, and likewise, the global version of the weak bisimilarity relation is equivalent to the weak bisimilarity relation on $\mathrm{Delay}(X + 1)$.

This implies that the global version of the extensional term precision semantics for $L^{gl}_{\mathcal{U}}X$ agrees with the corresponding notion for $\mathrm{Delay}(X+1)$. Then adequacy follows by proving the corresponding result for $\mathrm{Delay}(X + 1)$ which in turn follows from the definitions of the relations.

[6] A type X is clock-irrelevant if there is an isomorphism $\forall k.X \cong X$.

[7] The proof relies on the existence of an operation $\mathsf{force} : \forall k.\ \triangleright^k A \to \forall k.X$ that allows us to eliminate the later operator under a clock quantifier. This must be added as an axiom in guarded type theory.

7 DISCUSSION

7.1 Related Work

Eremondi [12] uses guarded type theory to define a syntactic model for a gradually-typed source language with dependent types. By working in guarded type theory, they are able to give an exact semantics to non-terminating computations in a language which is logically consistent, allowing for metatheoretic results about the source language to be proven. Similarly to our approach, they define a guarded lift monad to model potentially- nonterminating computations and use guarded recursion to model the dynamic type. However, they do not give a denotational semantics to term precision and it is unclear how to prove the gradual guarantee in this setting. The work includes a formalization of the syntactic model in Guarded Cubical Agda.

Siek and Chen [29] give a proof in Agda of graduality for an operational semantics. While they do not use the Guarded Cubical extension, they do use a guarded logic of propositions shallowly embedded in Agda. Our denotational approach requires full guarded type theory not just guarded logic. An advantage of the denotational approach is that it easily validates equational reasoning, not just graduality, and it is completely independent of any particular syntax of gradual typing.

7.2 Mechanization

In parallel with developing the theory discussed in this paper, we are mechanizing our results and developing a reusable framework for proving graduality in Guarded Cubical Agda [35]. As of this writing, the work is in progress, but we have constructed most parts of the concrete model discussed in Section 6. For instance, we have defined types for predomains, error domains, and their morphisms and relations, and we used the guarded features to define the guarded lift + error monad and the dynamic type.

We plan to formalize the construction of perturbations and quasi-representable relations, but we have yet to decide whether to follow the approach we take in this work and define the abstract notion of intensional model and formalize the constructions in that setting, and then apply those abstract constructions to the concrete model. Alternatively, it may be better from a mechanization standpoint to carry out those abstract constructions explicitly in the concrete model, i.e., our representation of objects in the concrete model of predomains would include a field for the perturbations and our notion of relations would include fields for the push-pull property and quasi-representability. We leave this investigation to future work.

Lastly, we plan to formalize the adequacy result discussed in Section 6.3. This will involve adding axioms about clock quantification as well as about the clock-irrelevance of booleans and natural numbers, since as of this writing these axioms are not built-in to Guarded Cubical Agda. As an experiment, we have formalized some basic results involving clocks as part of our development.

7.3 Comparison to Explicit Step-Indexing

Working internally to guarded type theory reduces the overhead of needing to carry around the step-indices in the proofs as is required when using explicit step-indexing. Additionally, the logical relations

constructed to prove graduality in prior work [24, 25, 28] suffer from technical complications of requiring two separate expression relations, one that counts steps on the left and the other on the right, and there is no analogue of this in our approach. However, using two expression relations allows some but not all transitivite reasoning of term precision to be recovered. In the future we aim to explore if this approach is feasible in guarded semantics.

7.4 Synthetic Ordering

A key to managing the complexity of our concrete construction is in using a *synthetic* approach to step-indexing rather than working analytically with presheaves. This has helped immensely in our ongoing mechanization in cubical Agda as it sidesteps the need to formalize these constructions internally. However, there are other aspects of the model, the bisimilarity and the monotonicity, which are treated analytically and are similarly tedious. It may be possible to utilize further synthetic techniques to reduce this burden as well, and have all type intrinsically carry a notion of bisimilarity and ordering relation, and all constructions to automatically preserve them. A synthetic approach to ordering is common in (non-guarded) synthetic domain theory and has also been used for synthetic reasoning for cost models [13, 15].

7.5 Future Work

In the future, we plan to apply our approach to give a denotational semantics for languages that feature higher-order state or runtime-extensible dynamic typing [32] as well as richer type disciplines such as gradual dependent types and effect systems.

REFERENCES

[1] Amal J. Ahmed. 2006. Step-Indexed Syntactic Logical Relations for Recursive and Quantified Types. In *Programming Languages and Systems, 15th European Symposium on Programming, ESOP 2006, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2006, Vienna, Austria, March 27-28, 2006, Proceedings (Lecture Notes in Computer Science, Vol. 3924)*, Peter Sestoft (Ed.). Springer, 69–83. https://doi.org/10.1007/11693024_6

[2] Andrew W. Appel and David McAllester. 2001. An indexed model of recursive types for foundational proof-carrying code. *ACM Trans. Program. Lang. Syst.* 23, 5 (sep 2001), 657–683. https://doi.org/10.1145/504709.504712

[3] Robert Atkey and Conor McBride. 2013. Productive Coprogramming with Guarded Recursion. In *Proceedings of the 18th ACM SIGPLAN International Conference on Functional Programming* (Boston, Massachusetts, USA) *(ICFP '13)*. Association for Computing Machinery, New York, NY, USA, 197–208. https://doi.org/10.1145/2500365.2500597

[4] Magnus Baunsgaard Kristensen, Rasmus Ejlers Mogelberg, and Andrea Vezzosi. 2022. Greatest HITs: Higher Inductive Types in Coinductive Definitions via Induction under Clocks. In *Proceedings of the 37th Annual ACM/IEEE Symposium on Logic in Computer Science* (Haifa, Israel) *(LICS '22)*. Association for Computing Machinery, New York, NY, USA, Article 42, 13 pages. https://doi.org/10.1145/3531130.3533359

[5] Lars Birkedal, Rasmus Ejlers Mogelberg, Jan Schwinghammer, and Kristian Stovring. 2011. First Steps in Synthetic Guarded Domain Theory: Step-Indexing in the Topos of Trees. In *2011 IEEE 26th Annual Symposium on Logic in Computer Science*. 55–64. https://doi.org/10.1109/LICS.2011.16

[6] Venanzio Capretta. 2005. General Recursion via Coinductive Types. *Logical Methods in Computer Science* Volume 1, Issue 2 (July 2005). https://doi.org/10.2168/LMCS-1(2:1)2005

[7] Matteo Cimini and Jeremy G. Siek. 2016. The Gradualizer: A Methodology and Algorithm for Generating Gradual Type Systems. In *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (St. Petersburg, FL, USA) *(POPL '16)*. Association for Computing Machinery, New York, NY, USA, 443–455. https://doi.org/10.1145/2837614.2837632

[8] Cyril Cohen, Thierry Coquand, Simon Huber, and Anders Mörtberg. 2017. Cubical type theory: A constructive interpretation of the univalence axiom. *IFCoLog J. Log. Appl.* (2017).

[9] Pierre-Louis Curien, Marcelo P. Fiore, and Guillaume Munch-Maccagnoni. 2016. A theory of effects and resources: adjunction models and polarised calculi. In *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2016, St. Petersburg, FL, USA, January 20 - 22, 2016*, Rastislav Bodík and Rupak Majumdar (Eds.). ACM, 44–56. https://doi.org/10.1145/2837614.2837652

[10] Brian Patrick Dunphy. 2002. *Parametricity As a Notion of Uniformity in Reflexive Graphs*. Ph. D. Dissertation. University of Illinois at Urbana-Champaign, Champaign, IL, USA. Advisor(s) Reddy, Uday.

[11] Jeff Egger, Rasmus Ejlers Møgelberg, and Alex Simpson. 2014. The enriched effect calculus: syntax and semantics. *J. Log. Comput.* 24, 3 (2014), 615–654. https://doi.org/10.1093/LOGCOM/EXS025

[12] Joseph S. Eremondi. 2023. *On the design of a gradual dependently typed language for programming*. Ph. D. Dissertation. University of British Columbia. https://doi.org/10.14288/1.0428823

[13] MARCELO P. FIORE. 1997. An enrichment theorem for an axiomatisation of categories of domains and continuous functions. *Mathematical Structures in Computer Science* 7, 5 (1997), 591–618. https://doi.org/10.1017/S0960129597002429

[14] Ronald Garcia, Alison M. Clark, and Éric Tanter. 2016. Abstracting Gradual Typing. In *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (St. Petersburg, FL, USA) *(POPL '16)*. Association for Computing Machinery, New York, NY, USA, 429–442. https://doi.org/10.1145/2837614.2837670

[15] Harrison Grodin, Yue Niu, Jonathan Sterling, and Robert Harper. 2024. Decalf: A Directed, Effectful Cost-Aware Logical Framework. *Proc. ACM Program. Lang.* 8, POPL (2024), 273–301. https://doi.org/10.1145/3632852

[16] Claudio Hermida, Uday S. Reddy, and Edmund P. Robinson. 2014. Logical Relations and Parametricity – A Reynolds Programme for Category Theory and Programming Languages. *Electronic Notes in Theoretical Computer Science* 303 (2014), 149–180. https://doi.org/10.1016/j.entcs.2014.02.008 Proceedings of the Workshop on Algebra, Coalgebra and Topology (WACT 2013).

[17] G. Janelidze and G.M. Kelly. 2001. A Note on Actions of a Monoidal Category. *Theory and Applications of Categories* 9, 4 (2001), 61–91.

[18] Paul Blain Levy. 1999. Call-by-Push-Value: A Subsuming Paradigm. In *Typed Lambda Calculi and Applications, 4th International Conference, TLCA'99, L'Aquila, Italy, April 7-9, 1999, Proceedings (Lecture Notes in Computer Science, Vol. 1581)*, Jean-Yves Girard (Ed.). Springer, 228–242. https://doi.org/10.1007/3-540-48959-2_17

[19] QingMing Ma and John C. Reynolds. 1991. Types, Abstractions, and Parametric Polymorphism, Part 2. In *Proceedings of the 7th International Conference on Mathematical Foundations of Programming Semantics*. Springer-Verlag, Berlin, Heidelberg, 1–40.

[20] Rasmus Ejlers Møgelberg and Marco Paviotti. 2016. Denotational Semantics of Recursive Types in Synthetic Guarded Domain Theory. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science* (New York, NY, USA) *(LICS '16)*. Association for Computing Machinery, New York, NY, USA, 317–326. https://doi.org/10.1145/2933575.2934516

[21] Rasmus Ejlers Møgelberg and Niccolò Veltri. 2019. Bisimulation as Path Type for Guarded Recursive Types. *Proc. ACM Program. Lang.* 3, POPL, Article 4 (jan 2019), 29 pages. https://doi.org/10.1145/3290317

[22] H. Nakano. 2000. A modality for recursion. In *Proceedings Fifteenth Annual IEEE Symposium on Logic in Computer Science (Cat. No.99CB36332)*. 255–266. https://doi.org/10.1109/LICS.2000.855774

[23] Georg Neis, Derek Dreyer, and Andreas Rossberg. 2009. Non-Parametric Parametricity. In *International Conference on Functional Programming (ICFP)*. 135–148. https://doi.org/10.1145/1596550.1596572

[24] Max S. New and Amal Ahmed. 2018. Graduality from Embedding-Projection Pairs. *Proc. ACM Program. Lang.* 2, ICFP, Article 73 (jul 2018), 30 pages. https://doi.org/10.1145/3236768

[25] Max S. New, Eric Giovannini, and Daniel R. Licata. 2023. Gradual Typing for Effect Handlers. *Proc. ACM Program. Lang.* 7, OOPSLA2 (2023), 1758–1786. https://doi.org/10.1145/3622860

[26] Max S. New, Dustin Jamner, and Amal Ahmed. 2019. Graduality and Parametricity: Together Again for the First Time. *Proc. ACM Program. Lang.* 4, POPL, Article 46 (dec 2019), 32 pages. https://doi.org/10.1145/3371114

[27] Max S. New and Daniel R. Licata. 2018. Call-by-name Gradual Type Theory. In *Formal Structures for Computation and Deduction, Oxford England*. https://doi.org/10.4230/LIPIcs.FSCD.2018.24

[28] Max S. New, Daniel R. Licata, and Amal Ahmed. 2019. Gradual Type Theory. *Proc. ACM Program. Lang.* 3, POPL, Article 15 (jan 2019), 31 pages. https://doi.org/10.1145/3290328

[29] Jeremy G. Siek and Tianyu Chen. 2021. Parameterized cast calculi and reusable meta-theory for gradually typed lambda calculi. *J. Funct. Program.* 31 (2021), e30. https://doi.org/10.1017/S0956796821000241

[30] Jeremy G. Siek and Walid Taha. 2006. Gradual Typing for Functional Languages. In *Scheme and Functional Programming Workshop (Scheme)*. 81–92.

[31] Jeremy G. Siek, Michael M. Vitousek, Matteo Cimini, and John Tang Boyland. 2015. Refined Criteria for Gradual Typing. In *1st Summit on Advances in Programming Languages (SNAPL 2015) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 32)*, Thomas Ball, Rastislav Bodik, Shriram Krishnamurthi, Benjamin S. Lerner, and Greg Morrisett (Eds.). Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 274–293. https://doi.org/10.4230/LIPIcs.SNAPL.2015.274
[32] Jonathan Sterling, Daniel Gratzer, and Lars Birkedal. 2022. Denotational semantics of general store and polymorphism. *CoRR* abs/2210.02169 (2022). https://doi.org/10.48550/ARXIV.2210.02169 arXiv:2210.02169
[33] Sam Tobin-Hochstadt and Matthias Felleisen. 2006. Interlanguage Migration: From Scripts to Programs. In *Dynamic Languages Symposium (DLS)*. 964–974.
[34] Sam Tobin-Hochstadt and Matthias Felleisen. 2008. The Design and Implementation of Typed Scheme. In *ACM Symposium on Principles of Programming Languages (POPL), San Francisco, California.*
[35] Niccolò Veltri and Andrea Vezzosi. 2020. Formalizing π-Calculus in Guarded Cubical Agda. In *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs* (New Orleans, LA, USA) *(CPP 2020)*. Association for Computing Machinery, New York, NY, USA, 270–283. https://doi.org/10.1145/3372885.3373814

A GRADUAL TYPING SYNTAX

Theorem A.1. *For every A, there is a derivation $dyn(A) : A \sqsubseteq D$*

Proof. By induction on A:

(1) $A = ?$, then $r(?) :? \sqsubseteq ?$
(2) $A = \text{Nat}$, then $\text{Inj}_{\mathbb{N}} : \text{Nat} \sqsubseteq ?$
(3) $A = A_1 \rightharpoonup A_2$ then $(\text{dyn}(A_1) \rightharpoonup \text{dyn}(A_2))\text{Inj}_{\rightarrow} : A_1 \rightharpoonup A_2 \sqsubseteq ?$
(4) $A = A_1 \times A_2$ then $(\text{dyn}(A_1) \times \text{dyn}(A_2))\text{Inj}_{\times} : A_1 \times A_2 \sqsubseteq ?$

□

Theorem A.2. *For any two derivations $c, c' : A \sqsubseteq A'$ of the same precision $c \equiv c'$*

Proof. (1) We show this by showing that derivations have a canonical form.
The following presentation of precision derivations has unique derivations

$$\text{refl}(D) : D \sqsubseteq D \qquad \text{Inj}_{\text{nat}} : \text{Nat} \sqsubseteq D$$

$$\text{refl}(\text{Nat}) : \text{Nat} \sqsubseteq \text{Nat} \qquad \frac{c : A_i \rightharpoonup A_o \sqsubseteq D \rightharpoonup D}{c(\text{Inj}_{\text{arr}}) : A_i \rightharpoonup A_o \sqsubseteq \text{Nat}}$$

$$\frac{c : A_i \sqsubseteq A_i' \qquad d : A_o \sqsubseteq A_o'}{c \rightharpoonup d : A_i \rightharpoonup A_o \sqsubseteq A_i' \rightharpoonup A_o'}$$

Since it satisfies reflexivity, cut-elimination and congruence, it is a model of the original theory. Since it is a sub-theory of the original theory, it is equivalent.

□

Type precision is a binary relation on typed terms. The original gradual guarantee rules are as follows:

$$\frac{\Gamma^{\sqsubseteq} \vdash M \sqsubseteq M' : c \qquad c : A \sqsubseteq A' \qquad c' : A \sqsubseteq A_2'}{\Gamma^{\sqsubseteq} \vdash M \sqsubseteq (M :: A_2') : c'}$$

$$\frac{\Gamma^{\sqsubseteq} \vdash M \sqsubseteq M' : c \qquad c : A \sqsubseteq A' \qquad c' : A_2 \sqsubseteq A'}{\Gamma^{\sqsubseteq} \vdash (M :: A_2) \sqsubseteq M' : c'}$$

Where the cast $M :: A_2$ is defined to be

$$\text{dn}\, dyn(A_2)\, \text{up}\, dyn(A)\, M$$

These two rules are admissible in our presentation.

For the first rule, we first prove that $\text{dn}\, dyn(A_2)\, \text{up}\, dyn(A)\, M = \text{dn}\, c'\, \text{up}\, c\, M$

$$\begin{aligned}
\text{dn}\, dyn(A_2)\, \text{up}\, dyn(A)\, M &= \text{dn}\, (c\text{dyn}(A'))\, \text{up}\, (c'\text{dyn}(A'))\, M && \text{(All derivations are equal)} \\
&= \text{dn}\, c\, \text{dn}\, \text{dyn}(A')\, \text{up}\, \text{dyn}(A')\, \text{up}\, c'\, M && \text{(functoriality)} \\
&= \text{dn}\, c\, \text{up}\, c'\, M && \text{(retraction)}
\end{aligned}$$

Then the rest follows by the up/dn rules above and the fact that precision derivations are all equal.

Thus the following properties are sufficient to provide an extensional model of gradual typing without requiring transitivity of term precision:

(1) Every precision is representable in the above sense,
(2) The association of casts to precision is functorial
(3) Type constructors are covariant functorial with respect to relational identity and composition

B CALL-BY-PUSH-VALUE

B.1 Morphisms of CBPV Models

There are two relevant notions of *morphism* of CBPV models: *strict* and *lax*. Given call-by-push-value models $\mathcal{M}_1 = (\mathcal{V}_1, \mathcal{E}_1, \rightarrow_1, U_1, F_1)$ and $\mathcal{M}_2 = (\mathcal{V}_2, \mathcal{E}_2, \rightarrow_2, U_2, F_2)$, A *strict* morphism G from M_1 to M_2 is given by a pair of functors $G_{\mathcal{V}} : V_1 \rightarrow V_2$ and $G_{\mathcal{E}} : E_1 \rightarrow E_2$ that strictly presere the constructors:

(1) $G_{\mathcal{E}} \circ F_1 = F_2 \circ G_{\mathcal{V}}$
(2) $G_{\mathcal{V}} \circ U_1 = U_2 \circ G_{\mathcal{E}}$
(3) $G_{\mathcal{E}}(A \rightarrow_1 B) = G_{\mathcal{V}}(A) \rightarrow_2 G_{\mathcal{E}}(B)$
(4) $G_{\mathcal{V}}(A_1 \times_1 A_2) = G_{\mathcal{V}}(A_1) \times_2 G_{\mathcal{V}}(A_2)$
(5) $G_{\mathcal{V}}(1_1) = 1_2$

As well as strictly preserving the corresponding universal morphisms and coherence isomorphisms.

A lax morphism instead preserves these only up to transformation

(1) $G_{\mathcal{E}} \circ F_1 \Rightarrow F_2 \circ G_{\mathcal{V}}$
(2) $G_{\mathcal{V}} \circ U_1 \Rightarrow U_2 \circ G_{\mathcal{E}}$
(3) $G_{\mathcal{E}}(A \rightarrow_1 B) \Rightarrow G_{\mathcal{V}}(A) \rightarrow_2 G_{\mathcal{E}}(B)$
(4) $G_{\mathcal{V}}(A_1 \times_1 A_2) \Rightarrow G_{\mathcal{V}}(A_1) \times_2 G_{\mathcal{V}}(A_2)$
(5) $G_{\mathcal{V}}(1_1) \Rightarrow 1_2$

Additionally a lax morphism should have a relationship between these transformations and the universal morphisms, but we will only consider lax morphisms of thin categories, where such conditions hold trivially.

B.2 Kleisli Actions of CBPV Type Constructors

In CBPV models, all the type constructors are interpreted as functors:

(1) $\rightarrow : \mathcal{V}^{\text{op}} \times \mathcal{E} \rightarrow \mathcal{E}$
(2) $\times : \mathcal{V} \times \mathcal{V} \rightarrow \mathcal{V}$
(3) $F : \mathcal{V} \rightarrow \mathcal{E}$
(4) $U : \mathcal{E} \rightarrow \mathcal{V}$

That is, they all have functorial actions on *pure* morphisms of value types and *linear* morphisms of computation types. We use these functorial actions extensively in the construction of casts and their corresponding perturbations. But when defining downcasts of value types and upcasts of computation types, we additionally need a second functorial action of these categories: functoriality in *impure* morphisms of value types and *non-linear* morphisms of computation types. These notions of morphism are given by the *Kleisli* categories $\mathcal{V}_k$ and $\mathcal{E}_k$ which have value types and computation types as objects but morphisms are defined as

$$\mathcal{V}_k(A, A') = \mathcal{E}(FA, FA')$$
$$\mathcal{E}_k(B, B') = \mathcal{V}(UB, UB')$$

with composition given by composition in $\mathcal{E}/\mathcal{V}$. That is we need to define a second functorial action, that agrees with the above on objects for these Kleisli categories:

(1) $\overset{k}{\to}: \mathcal{V}_k{}^{\mathrm{op}} \Box \mathcal{E}_k \to \mathcal{E}_k$
(2) $\overset{k}{\times}: \mathcal{V}_k \Box \mathcal{V}_k \to \mathcal{V}_k$
(3) $\overset{k}{F}: \mathcal{V}_k \to \mathcal{E}_k$
(4) $\overset{k}{U}: \mathcal{E}_k \to \mathcal{V}_k$

Note that rather than the product of categories we use the "funny tensor product" $\Box$. This is because the action on impure/non-linear morphisms for $\overset{k}{\to}$ /$\overset{k}{\times}$ do not satisfy "joint functoriality" but instead only "separate functoriality", meaning we give rather than an action on morphisms in both categories simultaneously instead an action on each argument categories morphisms with the object in the other category fixed. The existence of these functorial actions for $\overset{k}{\to}$ and $\overset{k}{\times}$ is reliant on the *strength* of the adjunction. We describe them using the internal language of CBPV in order to more easily verify their existence/functoriality:

(1) For $\overset{k}{\to}$ we define for $\phi : \mathcal{E}(FA, FA')$ and $B \in \mathcal{E}$ the morphism $\phi \overset{k}{\to} B : \mathcal{V}(U(A' \to B), U(A \to B))$ as

$$t : U(A' \to B) \vdash \phi \overset{k}{\to} B = \{\lambda x.x' \leftarrow \phi\,[\mathrm{ret}x]; !tx'\} : U(A \to B)$$

and for $A \in \mathcal{V}$ and $f : \mathcal{V}(UB, UB')$ we define $A \overset{k}{\to} f : \mathcal{V}(U(A \to B), U(A \to B'))$ as

$$t : U(A \to B) \vdash A \overset{k}{\to} f = \{\lambda x.!f[\{!tx\}]\}$$

(2) For $\overset{k}{\times}$ we define for $\phi : \mathcal{E}(FA_1, FA_2)$ and $A' \in \mathcal{V}$ the morphism $\phi \overset{k}{\times} A_2$ as

$$\bullet : F(A_1 \times A_2) \vdash \phi \overset{k}{\times} A_2 = (x_1, x_2) \leftarrow \bullet; x_1' \leftarrow \phi[\mathrm{ret}x_1]; \mathrm{ret}(x_1', x_2) : F(A_1' \times A_2)$$

and $A_1 \overset{k}{\times} \phi$ is defined symmetrically.

(3) For $\overset{k}{U}$ we need to define for $f : \mathcal{V}(UB, UB')$ a morphism $\overset{k}{U}\, f : \mathcal{E}(FUB, FUB')$. This is simply given by the functorial action of F: $\overset{k}{U}\, f = F(f)$

(4) Similarly $\overset{k}{F}\, \phi = U\phi$

Functoriality in each argument is easily established, meaning for example for the function type is functorial in each argument:

(1) $(\phi \circ \phi') \overset{k}{\to} B = (\phi' \overset{k}{\to} B) \circ (\phi \overset{k}{\to} B)$
(2) $\mathsf{id} \overset{k}{\to} B = \mathsf{id}$
(3) $A \overset{k}{\to} (f \circ f') = (A \overset{k}{\to} f) \circ (A \overset{k}{\to} f)$
(4) $A \overset{k}{\to} \mathsf{id} = \mathsf{id}$

Finally, note that all of these constructions lift to squares in a double CBPV model since the squares themselves form a CBPV model and the projection functions preserve CBPV structure. For instance, given a square $\alpha : \phi \sqsubseteq^{Fc_i}_{Fc_o} \phi'$ and a horizontal morphism $d : B \circ\!\!-\!\!\bullet B'$ of appropriate type, we get a square

$$\alpha \overset{k}{\to} d : \phi \overset{k}{\to} B \sqsubseteq^{U(c_i \to d)}_{U(c_o \to d)} \phi' \overset{k}{\to} B'$$

C DETAILS OF THE CONSTRUCTION OF AN EXTENSIONAL MODEL

In Section 5.3, we outline the construction of an extensional model of gradual typing starting from a step-1 intensional model. In this section, we provide the details for each of the constructions mentioned there.

C.1 Constructing a Model with Perturbations

The goal of this section is to prove the following lemma:

Lemma C.1. *Let $\mathcal{M}$ be a step-1 intensional model. Then we can construct a step-2 intensional model.*

We begin with a definition and some lemmas that will be useful in the construction of the model.

Definition C.2. Let $c : A \circ\!\!-\!\!\bullet A'$ be a value relation of $\mathcal{M}$. Let P_A be a monoid of perturbations on A and $P_{A'}$ a monoid of perturbations on A'. A *push-pull structure* $\Pi^{\mathcal{V}}_c$ for c with respect to P_A and $P_{A'}$ consists of:

- A function push : $P_A \to P_{A'}$ such that for all $\delta^l \in P_A$ we have $\delta^l \sqsubseteq^c_c \mathrm{push}(\delta^l)$.
- A function pull : $P_{A'} \to P_A$ such that for all $\delta_r \in P_{A'}$ we have $\mathrm{pull}(\delta^r) \sqsubseteq^c_c \delta^r$.

We define a push-pull structure $\Pi^{\mathcal{E}}_d$ for $d : B \circ\!\!-\!\!\bullet B'$ with respect to perturbation monoids P_B and $P_{B'}$ in an analogous manner.

Lemma C.3. *Let $c : A \circ\!\!-\!\!\bullet A'$ and $c' : A' \circ\!\!-\!\!\bullet A''$ be value relations, and let $P_A, P_{A'}$ and $P_{A''}$ be monoids of perturbations. Given a push-pull structure $\Pi^{\mathcal{V}}_c$ for c with respect to P_A and $P_{A'}$, and and a push-pull structure $\Pi^{\mathcal{V}}_{c'}$ for c' with respect to $P_{A'}$ and $P_{A''}$, we can define a push-pull structure $\Pi^{\mathcal{V}}_{c \bullet c'}$ for $c \bullet c'$.*

Likewise, we can define a push-pull structure for the composition of computation relations.

Proof. We define $\Pi^{\mathcal{V}}_{c \bullet c'}$ as the following push-pull structure:

- $\mathrm{push}_{c \bullet c'} = \mathrm{push}_{c'} \circ \mathrm{push}_c$
- $\mathrm{pull}_{c \bullet c'} = \mathrm{pull}_c \circ \mathrm{pull}_{c'}$

We observe that the required squares exist for both push and pull. In particular, for push we have that $\delta^l \sqsubseteq^c_c \mathrm{push}_c(\delta^l)$ using the push property for c, and then using the push property for c' we have $\mathrm{push}_c(\delta^l) \sqsubseteq^{c'}_{c'} \mathrm{push}_{c'}(\mathrm{push}_c(\delta^l))$. We can then compose these squares horizontally to obtain the desired square. The pull property follows similarly.

The push-pull structure for the composition of computation relations is defined analogously. □

Lemma C.4. *Let $c : A \circ\!\!-\!\!\bullet A'$ and let P_A and $P_{A'}$ be monoids of perturbations. Given a push-pull structure $\Pi_c^{\mathcal{V}}$ for c with respect to P_A and $P_{A'}$, we can define a push-pull structure $\Pi_{Fc}^{\mathcal{E}}$ for $F(c)$ with respect to $\mathbb{N} \times P_A$ and $\mathbb{N} \times P_{A'}$.*

Likewise, from a push-pull structure for d with respect to P_B and $P_{B'}$, we can define a push-pull structure for Ud with respect to $\mathbb{N} \times P_B$ and $\mathbb{N} \times P_{B'}$.

Proof. We define $\mathrm{push}_{Fc} : \mathbb{N} \times P_A \to \mathbb{N} \times P_{A'}$ by

$$\mathrm{push}_{Fc}(n, a) = (n, \mathrm{push}_c(a)).$$

Then we need to build the following square:

$$\begin{array}{ccc} FA & \xrightarrow{Fc} & FA' \\ {\scriptstyle (\delta^*_{FA})^n \circ \mathrm{ptb}_A(a)}\downarrow & & \downarrow{\scriptstyle (\delta^*_{FA'})^n \circ \mathrm{ptb}_{A'}(\mathrm{push}_c(a))} \\ FA & \xrightarrow{Fc} & FA' \end{array}$$

This is obtained by pasting n copies of the square $\delta^*_{FA} \sqsubseteq^{Fc}_{Fc} \delta^*_{FA'}$ on top of the square corresponding to the push property for c.

The proof for Ud is analogous. □

Lemma C.5. *Let $c_1 : A_1 \circ\!\!-\!\!\bullet A_1'$ and $c_2 : A_2 \circ\!\!-\!\!\bullet A_2'$, and let $P_{A_1}, P_{A_2}, P_{A_1'}$ and $P_{A_2'}$ be monoids of perturbations. Given push-pull structures $\Pi_{c_1}^{\mathcal{V}}$ with respect to P_{A_1} and $P_{A_1'}$, and $\Pi_{c_2}^{\mathcal{V}}$ with respect to P_{A_2} and $P_{A_2'}$, we can define a push-pull structure $\Pi_{c_1 \times c_2}^{\mathcal{V}}$ with respect to $P_{A_1} \times P_{A_2}$ and $P_{A_1'} \times P_{A_2'}$.*

Proof. We define the push function for $\Pi_{c_1 \times c_2}^{\mathcal{V}}$ by $\mathrm{push}(a_1, a_2) = (\mathrm{push}_{c_1}(a_1), \mathrm{push}_{c_2}(a_2))$ and likewise for pull. The push property holds because it holds for each component. The pull property is established similarly. □

Lemma C.6. *Let $c : A \circ\!\!-\!\!\bullet A'$ and $d : B \circ\!\!-\!\!\bullet B'$. Given push-pull structures $\Pi_c^{\mathcal{V}}$ for c and $\Pi_d^{\mathcal{E}}$ for d, we can define a push-pull structure $\Pi_{c \to d}^{\mathcal{E}}$ for $c \to d$.*

Proof. Follows from the functorial action of $\to$ on perturbations and squares. □

We now proceed with the construction of the model:

Define a step-2 model $\mathcal{M}'$ as follows:

- Value objects are pairs consisting of:
  - A value object A in $\mathcal{V}_f$.
  - A monoid P_A of perturbations and a monoid homomorphism $\mathrm{ptb}_A : P_A \to \{f \in \mathcal{V}_f(A, A) \mid f \approx \mathrm{id}_A\}$.
- Computation objects are pairs consisting of:
  - A computation object B in $\mathcal{E}_f$.
  - A monoid P_B of perturbations and a monoid homomorphism $\mathrm{ptb}_B : P_B \to \{\phi \in \mathcal{E}_f(B, B) \mid \phi \approx \mathrm{id}_B\}$.
- Morphisms are given by morphisms of the underlying objects in $\mathcal{V}_f$ and $\mathcal{E}_f$, respectively.
- Given objects (A, P_A, ptb_A) and $(A', P_{A'}, \mathrm{ptb}_{A'})$ a value relation is a pair consisting of
  - A value relation $c \in \mathcal{V}_{sq}$.
  - A push-pull structure $\Pi_c^{\mathcal{V}}$ for c with respect to P_A and $P_{A'}$.
- Similarly, a computation relation between (B, P_B, ptb_B) and $(B', P_{B'}, \mathrm{ptb}_{B'})$ consists of
  - A computation relation $d \in \mathcal{E}_{sq}$.
  - A push-pull structure $\Pi_d^{\mathcal{E}}$ for d with respect to P_B and $P_{B'}$.
- The squares are the same as the squares of the original model $\mathcal{M}$
- We define composition of relations $(c, \Pi_c^{\mathcal{V}})$ and $(c', \Pi_{c'}^{\mathcal{V}})$ as $(c \bullet c', \Pi_{c \bullet c'}^{\mathcal{V}})$ where $\Pi_{c \bullet c'}^{\mathcal{V}}$ is as in Lemma C.3, and likewise for computation relations.

Now we define the actions of the functors:

- We define $\times$ on objects by

$$(A_1, P_{A_1}, \mathrm{ptb}_{A_1}) \times (A_2, P_{A_2}, \mathrm{ptb}_{A_2}) = (A_1 \times A_2, P_{A_1} \times P_{A_2}, \mathrm{ptb}_{A_1 \times A_2})$$

  where $\mathrm{ptb}_{A_1 \times A_2}(p_1, p_2) = \mathrm{ptb}_{A_1}(p_1) \times \mathrm{ptb}_{A_2}(p_2)$.
  Using Lemma C.5, we define $\times$ on relations by

$$(c_1, \Pi_{c_1}^{\mathcal{V}}), (c_2 \Pi_{c_2}^{\mathcal{V}}) = (c_1 \times c_2, \Pi_{c_1 \times c_2}^{\mathcal{V}}).$$

- We define F on objects by

$$F(A, P_A, \mathrm{ptb}_A) = (FA, \mathbb{N} \times P_A, \mathrm{ptb}_{FA}),$$

  where we define $\mathrm{ptb}_{FA}(n, a) = (\delta^*_{FA})^n \circ F(\mathrm{ptb}_A(a))$.
  Using Lemma C.4, we define F on relations by

$$F(c, \Pi_c^{\mathcal{V}}) = (Fc, \Pi_{Fc}^{\mathcal{E}}).$$

- We define U on objects by

$$U(B, P_B, \mathrm{ptb}_B) = (UB, \mathbb{N} \times P_B, \mathrm{ptb}_{UB}),$$

  where we define $\mathrm{ptb}_{UB}(n, b) = (\delta^*_{UB})^n \circ U(\mathrm{ptb}_B(b))$.
  Using Lemma C.4, we define U on relations by

$$U(d, \Pi_d^{\mathcal{E}}) = (Ud, \Pi_{Ud}^{\mathcal{V}}).$$

  We define $\to$ on objects by

$$(A, P_A, \mathrm{ptb}_A) \to (B, P_B, \mathrm{ptb}_B) = (A \to B, P_A^{op} \times P_B, \mathrm{ptb}_{A \to B}),$$

  where we define $\mathrm{ptb}_{A \to B}(a, b) = \mathrm{ptb}_A(a) \to \mathrm{ptb}_B(b)$.
  Using Lemma C.6, we define $\to$ on relations by

$$(c, \Pi_c^{\mathcal{V}}) \to (d, \Pi_d^{\mathcal{E}}) = (c \to d, \Pi_{c \to d}^{\mathcal{E}}).$$

We define a Kleisli arrow operation on perturbations that takes a perturbation in P_{UB} to a perturbation in $P_{U(A \to B)}$, as follows: Given a perturbation $u = (n, b) \in P_{UB}$ we define $\mathrm{id} \xrightarrow{\mathrm{k}} u$ to be $(n, \mathrm{id}_{P_A}, b) \in P_{U(A \to B)}$. We need to show that

$$\mathrm{ptb}_{U(A \to B)}(\mathrm{id} \xrightarrow{\mathrm{k}} (n, b)) = \mathrm{id} \xrightarrow{\mathrm{k}} \mathrm{ptb}_{UB}(n, b).$$

That is, we need to show that

$$(\delta^*_{U(A \to B)})^n \circ U(\mathrm{id}_A \to \mathrm{ptb}_B(b)) = \mathrm{id} \xrightarrow{\mathrm{k}} (\delta^*_{UB})^n \circ U(\mathrm{ptb}_B(b))$$

This can be checked by unfolding the definition of $\xrightarrow{\mathrm{k}}$; the proof uses the fact that δ^*_{UB} commutes with computation morphisms.

Given a perturbation $q = (n, a) \in P_{FA}$ we define $(n, a) \overset{\mathrm{k}}{\to} \mathrm{id} = (n, a, \mathrm{id}_{P_B}) \in P_{U(A \to B)}$. We need to show that

$$\mathrm{ptb}_{U(A\to B)}((n, a) \overset{\mathrm{k}}{\to} \mathrm{id}) = \mathrm{ptb}_{FA}(n, a) \overset{\mathrm{k}}{\to} \mathrm{id}.$$

By similar reasoning to the above, this holds because δ^*_{FA} commutes with computation morphisms.

We can similarly construct a Kleisli product operation on perturbations. Given a perturbation $(n, a_1, a_2) \in P_{F(A_1 \times A_2)}$ we define $(n, a_1, a_2) \overset{\mathrm{k}}{\times} \mathrm{id} = (n, a_1, \mathrm{id}_{P_{A_2}})$, and likewise we define $\mathrm{id} \overset{\mathrm{k}}{\times} (n, a_1, a_2) = (n, \mathrm{id}_{P_{A_1}}, a_2)$.

C.2 Constructing a Model with Quasi-Representable Relations

The goal of this section is to prove the following lemma:

Lemma C.7. *Let $\mathcal{M}$ be a step-2 intensional model. Then we can construct a step-3 intensional model.*

Before proceeding with the proof, we begin with a definition.

Definition C.8 (representation structure). Let $c : A \circ\!\!-\!\!\bullet A'$ be a value relation. A *left-representation structure* ρ^L_c for c consists of a value morphism $e_c \in \mathcal{V}_f(A, A')$ such that c is quasi-left-representable by e_c (see Definition 5.4). We similarly define a right-representation structure ρ^R_c for c to consist of a morphism $p_c \in \mathcal{V}_f(A', A)$ such that c is quasi-right-representable by p_c (see Definition 5.5).

Likewise, let $d : B \circ\!\!-\!\!\bullet B'$ be a computation relation. Left- and right-representation structures for d are defined in a similar manner, except the representing morphisms are now computation morphisms rather than value morphisms.

Recall the notion of quasi-order-equivalence as defined in Definition 5.6. The following lemma will be useful in showing that two relations are quasi-order-equivalent.

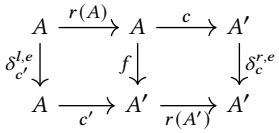
Lemma C.9. *Let $c, c' : A \circ\!\!-\!\!\bullet A'$. If c and c' are both quasi-left-representable by the same f, then $c \approx c'$.*

Dually, if d and d' are both quasi-right-representable by the same ϕ, then $d \approx d'$.

Proof. By UpR for c', there exists a perturbation $\delta^{l,e}_{c'}$ and a square $\mathrm{UpR}_{c'} : \delta^{l,e}_{c'} \sqsubseteq^{r(A)}_{c'} f$.

By UpL for c, there exists a perturbation $\delta^{r,e}_c$ and a square $\mathrm{UpL}_c : f \sqsubseteq^{c}_{r(A')} \delta^{r,e}_c$.

Composing these horizontally we get the following square:



And since $r(A) \bullet c = c$ and $c' \bullet r(A') = c'$, we are finished.

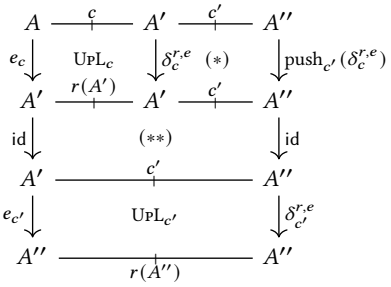
The other square (i.e., with c' on top) is constructed in an analogous manner. □

Lemma C.10. *In the below, let $c : A \circ\!\!-\!\!\bullet A'$ and $c' : A' \circ\!\!-\!\!\bullet A''$ and $d : B \circ\!\!-\!\!\bullet B'$ and $d' : B' \circ\!\!-\!\!\bullet B''$.*

(1) *Given left-representation structures ρ^L_c for c and $\rho^L_{c'}$ for c', we can define a left-representation structure for the composition $c \bullet c'$*
(2) *Given right-representation structures ρ^R_d for d and $\rho^R_{d'}$ for d', we can define a right-representation structure for the composition $d \bullet d'$*
(3) *Given right-representation structures ρ^R_{Fc} for Fc and $\rho^R_{Fc'}$ for Fc', we can define a right-representation structure $\rho^R_{F(c\bullet c')}$ for $F(c \bullet c')$.*
(4) *Given left-representation structures ρ^L_{Ud} for Ud and $\rho^L_{Ud'}$ for Ud', we can define a left-representation structure $\rho^L_{U(d\bullet d')}$ for $U(d \bullet d')$.*

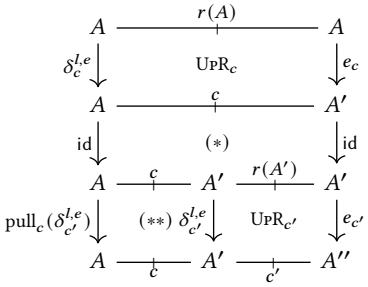
Proof. (1) We define $\rho^L_{c\bullet c'}$ as follows. In the definitions of the perturbations, we make use of the fact that c and c' satisfy the push-pull property.

- $e_{c\bullet c'} = e_{c'} \circ e_c$
- $\delta^{r,e}_{c\bullet c'} = \delta^{r,e}_{c'} \circ \mathrm{push}_{c'}(\delta^{r,e}_c)$
- $\delta^{l,e}_{c\bullet c'} = \mathrm{pull}_c(\delta^{l,e}_{c'}) \circ \delta^{l,e}_c$
- UpL is the following square:



The square $(*)$ exists by the push-pull property for c', and the square $(**)$ exists because r is a unit for horizontal composition, so $r(A') \bullet c' = c'$, and so this is simply the identity square $\mathrm{id}_{c'} \in \mathcal{V}_{sq}(c', c')$.

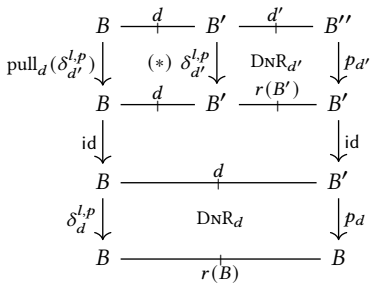
- UpR is the following square:



(2) We define $\rho^R_{d\bullet d'}$ as follows:

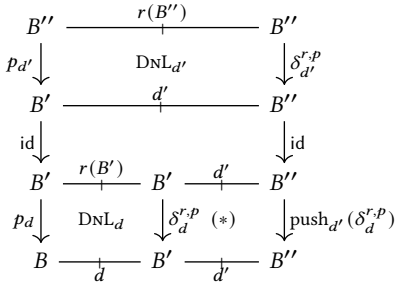
- $p_{d\bullet d'} = p_d \circ p_{d'}$
- $\delta^{l,p}_{d\bullet d'} = \delta^{l,p}_d \circ \mathrm{pull}_d(\delta^{l,p}_{d'})$
- $\delta^{r,p}_{d\bullet d'} = \mathrm{push}_{d'}(\delta^{r,p}_d) \circ \delta^{r,p}_{d'}$

- DnR is the following square:

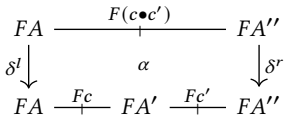


Here the square $(*)$ exists by the push-pull property for d.

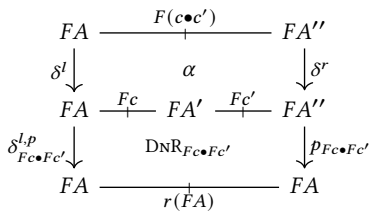
- DnL is the following square:



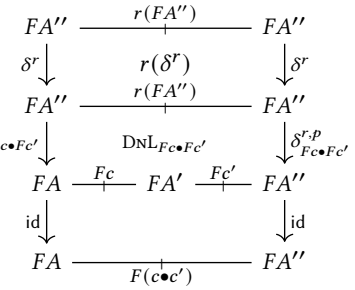
(3) We define $\rho^R_{F(c \bullet c')}$ as follows.
First, we claim that $F(c \bullet c')$ and $Fc \bullet Fc'$ are both quasi-left-represented by $Fe_{c'} \circ Fe_c$. Indeed, we have by part (1) that $e'_c \circ e_c$ left-represents $c \bullet c'$, and then since F preserves perturbations and squares it follows that $F(e'_c \circ e_c) = Fe_{c'} \circ Fe_c$ quasi-left-represents $F(c \bullet c')$. On the other hand, we also know that Fe_c quasi-left-represents Fc and $Fe_{c'}$ left-represents Fc', so their composition quasi-left-represents $Fc \bullet Fc'$. Thus, by Lemma C.9, there is a square α of the form



for some perturbations δ^l and δ^r. We define the projection $p_{F(c \bullet c')}$ to be $p_{Fc \bullet Fc'} \circ \delta^r$. We define $\delta^{l,p}_{F(c \bullet c')}$ to be $\delta^{l,p}_{Fc \bullet Fc'} \circ \delta^l$ Then we can build the square for DnR by pasting the square α on top of the DnR square for the composition $Fc \bullet Fc'$, as shown below:



We define $\delta^{r,p}_{F(c \bullet c')}$ to be $\delta^{r,p}_{Fc \bullet Fc'} \circ \delta^r$. For DnL, we paste the identity square $\delta^r \sqsubseteq \delta^r$ on top of the DnL square for the composition $Fc \bullet Fc'$, and below that we paste the square $\mathrm{id} \sqsubseteq^{Fc \bullet Fc'}_{F(c \bullet c')} \mathrm{id}$ which we get from the fact that F is lax.



(4) We define $\rho^L(U(d \bullet d'))$ in an analogous manner to the above.

□

Lemma C.11. *Let $c : A \circ\!\!-\!\!\bullet A'$, and let ρ^L_c be a left-representation structure for c. Then we can define a left-representation structure $\rho^L_{UF(c)}$ for $UF(c)$.*

Similarly, let $d : B \circ\!\!-\!\!\bullet B'$ and let ρ^R_d be a right-representation structure for d. Then we can define a right-representation structure $\rho^R_{FU(d)}$ for $FU(d)$.

Proof. We define $\rho^L_{UF(c)}$ as follows:

- $e_{UF(c)} = UF(e_c)$
- $\delta^{r,e}_{UF(c)} = UF(\delta^{r,e}_c)$ (which is in the monoid of perturbations of $UF(A')$ because the perturbation monoids are closed under the actions of the functors F and U)
- $\delta^{l,e}_{UF(c)} = UF(\delta^{l,e}_c)$
- We get the two commuting squares by the functorial action of UF on the two squares for c, i.e., $\mathrm{UpR}_{UF(c)} = UF(\mathrm{UpR}_c)$

We define $\rho^R_{FU(d)}$ in a similar manner. □

Lemma C.12. *Let $c_1 : A_1 \circ\!\!-\!\!\bullet A'_1$ and $c_2 : A_2 \circ\!\!-\!\!\bullet A'_2$. Let $\rho^L_{c_1}$ be a left-representation structure for c_1, and let $\rho^L_{c_2}$ be a left-representation structure for c_2. Then we can define a left-representation structure for $c_1 \times c_2$.*

Likewise, let $\rho^R_{Fc_1}$ and $\rho^R_{Fc_2}$ be right-representation structures for Fc_1 and Fc_2 respectively. Then we can define a right-representation structure for $F(c_1 \times c_2)$.

Proof. We define $\rho^L_{c_1 \times c_2}$ as follows:

- $e_{c_1 \times c_2} = e_{c_1} \times e_{c_2}$
- $\delta^{r,e}_{c_1 \times c_2} = \delta^{r,e}_{c_1} \times \delta^{r,e}_{c_2}$ and likewise for $\delta^{l,e}_{c_1 \times c_2}$
- We get the commuting squares via the functorial action of $\times$ on the corresponding squares for c_1 and c_2.

We define $\rho^R_{F(c_1 \times c_2)}$ as follows:

- $p_{F(c_1 \times c_2)} = (p_{Fc_1} \overset{\mathrm{k}}{\times} A_2) \circ (A'_1 \overset{\mathrm{k}}{\times} p_{Fc_2})$
- $\delta^{l,p}_{F(c_1 \times c_2)} = (\delta^{l,p}_{Fc_1} \overset{\mathrm{k}}{\times} A_2) \circ (A_1 \overset{\mathrm{k}}{\times} \delta^{l,p}_{Fc_2})$
- $\delta^{r,p}_{F(c_1 \times c_2)} = (\delta^{r,p}_{Fc_1} \overset{\mathrm{k}}{\times} A'_2) \circ (A'_1 \overset{\mathrm{k}}{\times} \delta^{r,p}_{Fc_2})$
- The commuting squares are obtained via the functorial action of $\overset{\mathrm{k}}{\times}$ on the squares for Fc_1 and Fc_2.

□

Lemma C.13. *Let $c : A \circ\!\!-\!\!\bullet\ A'$ and $d : B \circ\!\!-\!\!\bullet\ B'$. Let ρ^L_c be a left-representation structure for c, and let ρ^R_d be a right-representation structure for d. Then we can define a right-representation structure for $c \to d$.*

Likewise, let ρ^R_{Fc} be a right-representation structure for Fc, and let ρ^L_{Ud} be a left-representation structure for Ud. Then we can define a left-representation structure for $U(c \to d)$.

Proof. We define $\rho^R_{c\to d}$ as follows:

- $p_{c\to d} = e_c \to p_d \in \mathcal{E}_f(A' \to B', A \to B)$ (using the functorial action of $\to$ on morphisms).
- $\delta^{l,p}_{c\to d} = \delta^{l,e}_c \to \delta^{l,p}_d$
- $\delta^{r,p}_{c\to d} = \delta^{r,e}_c \to \delta^{r,p}_d$
- The squares DnR and DnL are obtained via the functorial action of $\to$, i.e., we define

$$\mathrm{DnR}_{c\to d} = \mathrm{UpR}_c \to \mathrm{DnR}_d : (\delta^{l,e}_c \to \delta^{l,p}_d) \sqsubseteq^{c\to d}_{r(A\to B)} (e_c \to p_d),$$

and

$$\mathrm{DnL}_{c\to d} = \mathrm{UpL}_c \to \mathrm{DnL}_d.$$

We define $\rho^L_{U(c\to d)}$ as follows:

- $e_{U(c\to d)} = (p_{Fc} \xrightarrow{\mathrm{k}} B') \circ (A \xrightarrow{\mathrm{k}} e_{Ud})$
- $\delta^{r,e}_{U(c\to d)} = (\delta^{r,p}_{Fc} \xrightarrow{\mathrm{k}} B') \circ (A' \xrightarrow{\mathrm{k}} \delta^{r,e}_{Ud})$
- $\delta^{l,e}_{U(c\to d)} = (\delta^{l,p}_{Fc} \xrightarrow{\mathrm{k}} B) \circ (A \xrightarrow{\mathrm{k}} \delta^{l,e}_{Ud})$
- The squares UpL and UpR are obtained via the functorial action of $\xrightarrow{\mathrm{k}}$. For instance, UpL is given by the following square:

$$\begin{array}{ccc}
U(A \to B) & \xrightarrow{U(c\to d)} & U(A' \to B') \\
{\scriptstyle A \xrightarrow{\mathrm{k}} e_{Ud}}\downarrow & {\scriptstyle \mathrm{id}_{Fc} \xrightarrow{\mathrm{k}} \mathrm{UpL}_{Ud}} & \downarrow{\scriptstyle A' \xrightarrow{\mathrm{k}} \delta^{r,e}_{Ud}} \\
U(A \to B') & \xrightarrow{U(c\to r(B'))} & U(A' \to B') \\
{\scriptstyle p_{Fc} \xrightarrow{\mathrm{k}} B'}\downarrow & {\scriptstyle \mathrm{DnL}_{Fc} \xrightarrow{\mathrm{k}} \mathrm{id}_{r(B')}} & \downarrow{\scriptstyle \delta^{r,p}_{Fc} \xrightarrow{\mathrm{k}} B'} \\
U(A' \to B') & \xrightarrow{U(r(A')\to r(B'))} & U(A' \to B')
\end{array}$$

The construction of UpR is similar. □

Now we can give the proof of the main lemma:
We define a step-3 model $\mathcal{M}'$ as follows:

- The objects of $\mathcal{M}'$ are defined to be the same as the objects of $\mathcal{M}$.
- The value and computation morphisms in $\mathcal{M}'$ are the same as those of $\mathcal{M}$.
- A value relation is defined to be a tuple $(c, \rho^L_c, \rho^R_{Fc})$ with
  - c a value relation in $\mathcal{M}$,
  - ρ^L_c a left-representation structure for c, and
  - ρ^R_{Fc} a right-representation structure for Fc
- Likewise, a computation relation is defined to be a tuple $(d, \rho^R_d, \rho^L_{Ud})$ with
  - d a computation relation in $\mathcal{M}$,
  - ρ^R_d a right-representation structure for d, and
  - ρ^L_{Ud} a left-representation structure for Ud.
- Morphisms of value relations (i.e., the value squares) are defined by simply ignoring the representation structures. That is, a morphism of value relations $\alpha \in \mathcal{V}'_{sq}((c, \rho^L_c, \rho^R_{Fc}), (c'\rho^L_{c'}, \rho^R_{Fc'}))$ is simply a morphism of value relations in $\mathcal{V}_{sq}(c, c')$. Likewise for computations.

We define horizontal composition of relations and squares as follows: Let $c : A \circ\!\!-\!\!\bullet\ A'$ and $c' : A' \circ\!\!-\!\!\bullet\ A''$. We define

$$((c, \rho^L_c, \rho^R_{Fc}) \bullet (c', \rho^L_{c'}, \rho^R_{Fc'})) = (c \bullet c', \rho^L_{c\bullet c'}, \rho^R_{F(c\bullet c')}),$$

and

$$((d, \rho^R_d, \rho^L_{Ud}) \bullet (d', \rho^R_{d'}, \rho^L_{Ud'})) = (d \bullet d', \rho^R_{d\bullet d'}, \rho^L_{U(d\bullet d')}),$$

where $\rho^L_{c\bullet c'}$, $\rho^R_{F(c\bullet c')}$, $\rho^R_{d\bullet d'}$, and $\rho^L_{U(d\bullet d')}$ are as defined in Lemma C.10.

Now we define the functors F, U, $\times$, and $\to$. On objects, the behavior is the same as the respective functors in $\mathcal{M}$. For relations, we define

$$F(c, \rho^L_c, \rho^R_{Fc}) = (Fc, \rho^R_{Fc}, \rho^L_{UF(c)}),$$

and

$$U(d, \rho^R_d, \rho^L_{Ud}) = (Ud, \rho^L_{Ud}, \rho^R_{FU(d)}),$$

where $\rho^L_{UF(c)}$ and $\rho^R_{FU(d)}$ are as defined in the proof of Lemma C.11.

We define

$$(c_1, \rho^L_{c_1}, \rho^R_{Fc_1}) \times (c_2, \rho^L_{c_2}, \rho^R_{Fc_2}) = (c_1 \times c_2, \rho^L_{c_1\times c_2}, \rho^R_{F(c_1\times c_2)}),$$

where $\rho^L_{c_1\times c_2}$ and $\rho^R_{F(c_1\times c_2)}$ are as defined in the proof of Lemma C.12.

Lastly, we define

$$(c, \rho^L_c, \rho^R_{Fc}) \to (d, \rho^R_d, \rho^L_{Ud}) = (c \to d, \rho^R_{c\to d}, \rho^L_{U(c\to d)}),$$

where $\rho^R_{c\to d}$ and $\rho^L_{U(c\to d)}$ are as defined in the proof of Lemma C.13.

We now establish the quasi-order-equivalence for the functors. We already showed that $U(d \bullet d') \approx U(d)U(d')$ and $F(c \bullet c'') \approx F(c)F(c')$ in the proof of Lemma C.10. The other two properties $(cc') \to (dd') \approx (c \to d)(c' \to d')$ and $(c_1c'_1) \times (c_2c'_2) \approx (c_1 \times c_2)(c'_1 \times c'_2)$ are proved similarly, noting in both cases that the both relations are quasi-represented by the same morphism.

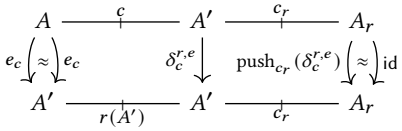
C.3 Constructing an Extensional Model

We aim to prove the following lemma:

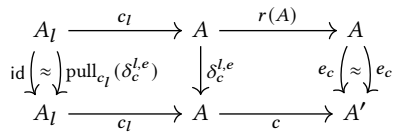
Lemma C.14. *Let $\mathcal{M}$ be a step-4 intensional model. Then we can define an extensional model.*

Proof. Recall the extensional model construction outlined in Section 5.3.3. We first establish the representability properties for this model. We show the left-representability squares; the right-representability squares are dual.

- We have the square



- We have the square



We observe that the functoriality of the CBPV connectives on relations up to order-equivalence is a direct consequence of the functoriality of the CBPV connectives on relations up to quasi-order-equivalence in the step-4 intensional model, since the perturbations on both sides of the square are by definition bisimilar to the identity.

Likewise, the retraction properties for the relations $\mathrm{Inj}_{\to} : U(D \to FD) \circ\!\!-\!\!\bullet D$, $\mathrm{Inj}_{\mathbb{N}} : \mathrm{Nat} \circ\!\!-\!\!\bullet D$, and $\mathrm{Inj}_{\times} : D \times D \circ\!\!-\!\!\bullet D$ hold because they held up to delays in the intensional model, and the delays disappear in the extensional model construction.

□