Logical Methods in Computer Science
Volume 16, Issue 1, 2020, pp. 7:1–7:45
https://lmcs.episciences.org/

Submitted Jan. 31, 2019
Published Jan. 31, 2020

CALL-BY-NAME GRADUAL TYPE THEORY

MAX S. NEW AND DANIEL R. LICATA

Northeastern University, Boston, USA
e-mail address: maxnew@ccs.neu.edu

Wesleyan University, Middletown, USA
e-mail address: dlicata@wesleyan.edu

Abstract. We present *gradual type theory*, a logic and type theory for call-by-name gradual typing. We define the central constructions of gradual typing (the dynamic type, type casts and type error) in a novel way, by universal properties relative to new judgments for *gradual type and term dynamism*. These dynamism judgments build on prior work in blame calculi and on the "gradual guarantee" theorem of gradual typing. Combined with the ordinary extensionality (η) principles that type theory provides, we show that most of the standard operational behavior of casts is *uniquely determined* by the gradual guarantee. This provides a semantic justification for the definitions of casts, and shows that non-standard definitions of casts must violate these principles. Our type theory is the internal language of a certain class of preorder categories called *equipments*. We give a general construction of an equipment interpreting gradual type theory from a 2-category representing non-gradual types and programs. This construction is a semantic analogue of the interpretation of gradual typing using contracts, and use it to build some concrete domain-theoretic models of gradual typing.

1. Introduction

Gradually typed languages allow for static and dynamic programming styles within the same language. They are designed with twin goals of allowing easy interoperability between static and dynamic portions of a codebase and facilitating a smooth transition from dynamic to static typing. This allows for the introduction of new typing features to legacy languages and codebases without the enormous manual effort currently necessary to migrate code

Key words and phrases: Gradual Typing, Type Systems, Program Logics, Category Theory, Denotational Semantics.

This material is based upon work supported by the National Science Foundation under grant CCF-1453796. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

This research was partially supported by the United States Air Force Research Laboratory under agreement numbers FA-95501210370 and FA-95501510053. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the United States Air Force Research Laboratory, the U.S. Government or Carnegie Mellon University.

DOI:10.23638/LMCS-16(1:7)2020

from a dynamically typed language to a fully statically typed language. Gradual typing allows exploratory programming and prototyping to be done in a forgiving, dynamically typed style, while later that code can be typed to ease readability and refactoring. Due to this appeal, there has been a great deal of research on extending gradual typing [37, 34] to numerous language features such as parametric polymorphism [21, 2, 16, 40], effect tracking [3], typestate [44], session types [15], refinement types [18] and security types [39]. Almost all work on gradual typing is based solely on operational semantics, and recent work such as [33] has codified some of the central design principles of gradual typing in an operational setting. In this paper, we are interested in complementing this operational work with a type-theoretic and category-theoretic analysis of these design principles. We believe this will improve our understanding of gradually typed languages, particularly with respect to principles for reasoning about program equivalence, and assist in designing and evaluating new gradually typed languages because it gives criteria for casts (that they form embedding-projection pairs) that imply graduality.

One of the central design principles for gradual typing is *gradual type soundness*. At its most general, this should mean that the types of the gradually typed language provide the same type-based reasoning that one could reasonably expect from a similar statically typed language, i.e. one with runtime errors and general recursion. While this has previously been defined using operational semantics and a notion of *blame* [42], the idea of soundness we consider here is that the types should provide the same extensionality (η) principles as in a statically typed language. This way, programmers can reason about the "typed" parts of gradual programs in the same way as in a fully static language. This definition fits nicely with a category-theoretic perspective, because the β and η principles correspond to definitions of connectives by a *universal property*.

The second design principle is the *gradual guarantee* [33], which we will refer to as *graduality* (by analogy with parametricity). Informally, graduality of a language means that syntactic changes from dynamic to static typing (or vice-versa) should result in simple, predictable changes to the semantics of a term. More specifically, if a portion of a program is made "more static"/"less dynamic" then the new program should either have the same behavior or result in a runtime type error. Other observable behavior such as values produced, I/O actions performed or termination should not be changed. In other words, a "less dynamic" program should expose "less information": by making types more static, we limit the interface for the program and thus hide behavior, replacing it with a runtime type error. Of course, limiting the interface is precisely what allows for the typed reasoning principles that gradual type soundness requires.

In this paper, we codify these two principles of soundness and graduality *directly* into a logical syntax we dub (call-by-name) *Gradual Type Theory* (Section 2). For graduality, we develop a logic of *type and term dynamism* that can be used to reason about the relationship between "more dynamic" and "less dynamic" versions of a program, and to give novel specifications/universal properties for the dynamic type, type errors, and runtime type casts of a gradually typed language. These universal properties extend the judgmental approach to type theory (see [20, 28]) to the key features of gradual typing. For soundness, we assert β and η principles as axioms of term dynamism, so that the logic models program behavior. Furthermore, using the η principles for types, we show that most of the operational rules of runtime casts of existing (call-by-name) gradually typed languages are *uniquely determined* by these constraints of soundness and graduality (Section 3). As an example application, uniqueness implies that a complicated space-efficient contract enforcement scheme in a

particular language (e.g. as in [35]) is equivalent to a standard wrapping implementation, *if* it satisfies soundness and graduality (which might be separately provable by a logical relations argument). Contrapositively, uniqueness implies that any enforcement scheme in a specific gradually typed language that is *not* equivalent to the standard "wrapping" ones *must* violate either soundness or graduality. We have chosen call-by-name because it is a simple setting with the necessary η principles (for negative types) to illustrate our technique. We have in other work considered application to other evaluation orders ([25]), which we discuss in more detail in Section 7.

We give a sound and complete category theoretic semantics for gradual type theory in terms of certain *preorder categories* (double categories where one direction is thin) (Section 4). We show that the contract interpretation of gradual typing [38] can be understood as a tool for constructing models (Section 5): starting from some existing language/category C, we first implement casts as suitable pairs of functions/morphisms in C, and then equip every type with canonical casts to the dynamic type. Technically, the first step forms a double category from a 2-category by interpreting vertical arrows as Galois insertions/coreflections, i.e., related pairs of an upcast and a downcast. Second, from a suitable choice of dynamic type, we construct a "vertical slice" preorder category whose objects are vertical arrows into the chosen dynamic type. We apply this to construct some concrete models in domains (Section 6).

Conceptually, gradual type theory is analogous to Moggi's *monadic metalanguage* [22]: it clarifies general principles present in many different programming languages; it is the internal language of a quite general class of category-theoretic structures; and, for a specific language, a number of useful results can be proved all at once by showing that a logical relation over it is a model of the type theory.

1.1. **A logic of dynamism and casts.** Before proceeding to the technical details, we explain at a high level how our type theory accounts for two key features of gradual typing: graduality and casts. The "gradual guarantee" as defined in [33] applies to a surface language where runtime type casts are implicitly inserted based on type annotations, but we will focus here on an analysis of fully elaborated languages, where explicit casts have already been inserted (so our work does not yet address gradual type checking). The gradual guarantee as defined in [33] makes use of a *syntactically less dynamic*[1] ordering on types: the dynamic type (universal domain) ? is the most dynamic, and A is less dynamic than B if B has the same structure as A but some sub-terms are replaced with ? (for example, $A \to (B \times C)$ is less dynamic than $? \to (B \times ?)$, $? \to ?$ and ?). Intuitively, a less dynamic type constrains the behavior of the program more, but consequently gives stronger reasoning principles. This notion is extended to closed well-typed *terms* $t : A$ and $t' : A'$ with A less dynamic than A': t is *syntactically less dynamic* than t' if t is obtained from t' by replacing the input and output type of each type cast with a less (or equally) dynamic type (in [33] this was called "precision"). For example, if $\mathsf{add1} : ? \to \mathbb{N}$ and $\mathsf{true} : ?$, then $\mathsf{add1}((? \Leftarrow \mathbb{N})(\mathbb{N} \Leftarrow ?)\mathsf{true})$ (cast true from dynamic to $\mathbb{N}$ and back, to assert it is a number) is syntactically less dynamic than $\mathsf{add1}((? \Leftarrow ?)(? \Leftarrow ?)\mathsf{true})$ (where both casts are the identity). Then the gradual guarantee [33] says that if t is syntactically less dynamic than t', then t is *semantically less dynamic* than t': either t evaluates to a type error (in which case t' may do anything) or

[1]Throughout this work, we will use the words "less than" to mean "less than or equal to" rather than "strictly less than", and similarly for other terms such as "greater than", "more dynamic", etc.

t, t' have the same first-order behavior (both diverge or both terminate with t producing a less dynamic value). In the above example, the less dynamic term always errors (because true fails the runtime $\mathbb{N}$ check), while the more dynamic term only errors if $\mathsf{add1}$ uses its argument as a number. In contrast, a program that returns a different value than $\mathsf{add1}(\mathsf{true})$ does will not be semantically less dynamic than it.

The approach we take in this paper is to give a *syntactic logic* for the *semantic* notion of one term being less dynamic than another, with $\mathcal{U}$ (type error) the least element, and all term constructors monotone. We call this the *term dynamism relation* $t \sqsubseteq t'$, and it includes not only syntactic changes in type casts, as above, but also equational laws like identity and composition for casts, and $\beta\eta$ rules—so $t \sqsubseteq t'$ intuitively means that t type-errors more than (or as much as) t', but is otherwise equal according to these equational laws. A programming language that is a model of our type theory will therefore be equipped with a semantic $t[\![\sqsubseteq]\!]t'$ relation validating these rules, so $t[\![\sqsubseteq]\!]t'$ if t type-errors more than t' up to these equational and monotonicity laws. In particular, making type cast annotations less dynamic will result in related programs, and if $[\![\sqsubseteq]\!]$ is adequate (i.e., no operationally distinguishable terms are order-equivalent), then this implies the gradual guarantee [33]. Therefore, we say a model of gradual type theory "satisfies graduality" in the same sense that we would say a language satisfies parametricity. We have developed operational models for CBV and CBPV languages that interpret the semantic ordering here as a type of contextual approximation [23, 25].

Next, we discuss the relationship between term dynamism and casts, the most novel aspect of our theory. Explicit casts in a gradually typed language are typically presented by the syntactic form $(B \Leftarrow A)t$, and their semantics is either defined by various operational reductions that inspect the structure of A and B, or by "contract" translations, which compile a language with casts to another language, where the casts are implemented as ordinary functions. In both cases, the behavior of casts is defined by inspection on types and part of the language definition, with little justification beyond intuition and precedent.

In gradual type theory, on the other hand, the behavior of casts is *not* defined by inspection of types. Rather, we use the new type and term dynamism judgments, which are defined *prior to* casts, to give a few simple and uniform rules specifying casts in all types via a universal property (optimal implementation of a specification). Our methodology requires isolating two special subclasses of casts, upcasts and downcasts. An upcast goes from a "more static" to a "more dynamic" type— for instance $(? \Leftarrow (A \to B))$ is an upcast from a function type up to the dynamic type—whereas a downcast is the opposite, casting to the more static type. We represent the relationship "A is less dynamic than B" by a *type dynamism* judgment $A \sqsubseteq B$ (which corresponds to the "naïve subtyping" of [42]). In gradual type theory, the upcast $\langle B \nwarrow A \rangle$ from A to B and the downcast $\langle A \swarrow B \rangle$ from B to A can be formed whenever $A \sqsubseteq B$. This leaves out certain casts like $(? \times \mathbb{N}) \Leftarrow (\mathbb{N} \times ?)$ where neither type is more dynamic than the other. However, as first recognized in [14], these casts are macro-expressible [9] as a composite of an upcast to the dynamic type and then a downcast from it (define $(B \Leftarrow A)t$ as the composite $\langle B \swarrow ? \rangle \langle ? \nwarrow A \rangle t$).

A key insight is that we can give upcasts and downcasts dual specifications using term dynamism, which say how the casts relate programs to type dynamism. If $A \sqsubseteq B$, then for any term $t : A$, the upcast $\langle B \nwarrow A \rangle t : B$ is the *least* dynamic term of type B that is more dynamic than t. In order-theoretic terms, $\langle B \nwarrow A \rangle t : B$ is the $\sqsubseteq$-meet of all terms $u : B$ with $t \sqsubseteq u$. Downcasts have a dual interpretation as a $\sqsubseteq$-join. Intuitively, this property

means upcast $\langle B \nwarrow A \rangle t$ behaves as much as possible like t itself, while supporting the additional interface provided by expanding the type from A to B.

This simple definition has powerful consequences that we explore in Section 3, because it characterizes the upcasts and downcasts up to program equivalence. We show that standard implementations of casts are the *unique* implementations that satisfy β, η and basic congruence rules. In fact, almost all of the standard operational rules of a simple call-by-name gradually typed language are term-dynamism equivalences in gradual type theory. The exception is rules that rely on disjointness of different type connectives (such as $\langle ? \to ? \swarrow ? \rangle \langle ? \nwarrow ? \times ? \rangle t \mapsto \mho$), which are independent, and can be added as axioms.

1.2. **Models of Gradual Typing.** In addition to axiomatizing graduality in gradual type theory, we also consider categorical semantics and denotational models of the theory.

We give a definition of a *model* of gradual type theory in *cartesian preorder multi-categories* (CPMS), which are related to categories internal to the category of preordered sets, i.e., sets with a reflexive, transitive relation. This presents a simple alternative, algebraic specification of type and term dynamism. A CPM is like a category where the sets of objects and arrows have the structure of a preorder, and the source, target, identity and composition functions are all monotone. However, as in type theory, and in contrast to categories, arrows can have 0 or more inputs rather than exactly 1. The ordering on objects models type dynamism and the ordering on terms models term dynamism, and the rest of the requirements succinctly describe the relationship between those two notions.

To model the casts, we in addition need that for any two objects with $A \sqsubseteq B$, there exist morphisms $A \to B$ and $B \to A$ that model upcasts and downcasts. In the category theory literature, this structure is called an *equipment* and we adapt existing constructions and results from that work [32].

We then prove an *initiality* theorem for gradual type theory with respect to the category of models, extending the classical correspondence between simply typed lambda calculus and cartesian closed categories [17] to gradual typing. In logical terms, we show that gradual type theory is *sound* and *complete* with respect to our notion of model. Of course this is no accident, we used this notion of model as the basis for our design of gradual type theory. However, we prefer to present the syntax first, since it does not require any knowledge of category theory to understand.

In addition to providing a different perspective on the structure of type and term dynamism, the CPM semantics of gradual typing enables us to systematically build models of gradual typing. In particular, we present the "contract interpretation" of casts as a semantic construction of a model of gradual typing from a cartesian 2-category. Furthermore we can decompose this construction into simple pieces. First, we form a double category from a 2-category by interpreting vertical arrows as Galois insertions/coreflections, i.e., related pairs of an upcast and a downcast. Second, from a suitable choice of dynamic type, we construct a "vertical slice" CPM whose objects are vertical arrows into the chosen dynamic type.

Finally, we instantiate this construction with several concrete models, and in doing so make a formal connection between gradual typing and domain-theoretic interpretations of dynamic typing. Such a connection has been folklore since the earliest days of higher order contracts, and we make this precise by constructing models of gradual type theory. First, we give a simple first-order model that doesn't support function types, but as a consequence is also elementary in that it only requires a solution to a covariant fixed point equation.

Then, we show that Dana Scott's classical construction of a model of types from retracts of a universal domain is an instance of our contract construction, but is *inadequate* for interpreting gradual typing because it conflates type errors and nontermination. Finally we show that a better model can be constructed by using a category of "ordered domains" that in addition to the domain ordering have a separate "type error ordering" that models term dynamism.

1.3. **Overview.** The paper proceeds as follows

- In Section 2 we present the syntax of gradual type theory (GTT).
- In Section 3, we formulate and prove many theorems *within* GTT, including many equivalences between casts.
- In Section 4, we define models of gradual type theory and prove that the syntax of GTT provides the initial model.
- In Section 5, we present a semantic formulation of the "contract interpretation" of gradual types as constructing a model of GTT from a suitable 2-category.
- In Section 6, we instantiate our contract interpretation with several concrete models, including classic domain-theoretic interpretations of dynamically typed lambda calculus.
- Finally, in Section 7, we discuss related work.

This article is an extended version of [24]. The primary differences are that we (1) expanded Section 3 to include proofs, illustrating the process of reasoning about gradually typed programs in the logic of GTT, (2) expanded Section 4 to include additional details of the initiality theorems, and (3) expanded Section 6 to more fully describe the construction of our denotational models.

While Sections 4, 5, 6 heavily use category-theoretic and domain-theoretic terminology and techniques, Sections 2 and 3 are self-contained presentations of the syntax and derivable theorems of GTT that require no knowledge of category or domain theory, so readers with an interest in gradual typing but not these semantic techniques may prefer to focus on these sections on GTT syntax.

2. Gradual Type Theory

In this section, we present the rules of gradual type theory (GTT). Gradual type theory presents the types, connectives and casts of gradual typing in a modular, type-theoretic way: the dynamic type, type error and casts are defined by rules using the *judgmental structure* of the type theory, specifically the new judgments for type and term dynamism which we add to the usual judgmental structure of typed lambda calculus. Since the judgmental structure is so important, we first present a bare-bones type theory we call *preorder type theory* (PTT) which only has base types. We can then modularly define what it means for this theory to have a dynamic type, type errors, casts, functions and products. Then gradual type theory is defined to be preorder type theory with all of these constructions.

2.1. **Preorder Type Theory.** Preorder type theory (PTT) has 6 judgments: types, contexts, type dynamism, dynamism contexts, terms and term dynamism. Their presuppositions (one is only allowed to make a judgment when these conditions hold) are presented in Figure 1, where A type and Γ ctx have no conditions. The types, contexts and terms (Figure 2) are structured as a standard type theory. Terms are treated as intrinsically typed with

$$\frac{A\,\text{type} \qquad A'\,\text{type}}{A \sqsubseteq A'} \qquad \frac{\Gamma\,\text{ctx} \qquad \Gamma'\,\text{ctx}}{\Phi : \Gamma \sqsubseteq \Gamma'} \qquad \frac{\Gamma\,\text{ctx} \qquad A\,\text{type}}{\Gamma \vdash t : A} \qquad \frac{\Phi : \Gamma \sqsubseteq \Gamma' \qquad A \sqsubseteq A' \qquad \Gamma \vdash t : A \qquad \Gamma' \vdash t' : A'}{\Phi \vdash t \sqsubseteq t' : A \sqsubseteq A'}$$

Figure 1: Judgment Presuppositions of Preorder Type Theory

$$\frac{X \in \Sigma_0}{X\,\text{type}} \qquad \frac{}{\cdot\,\text{ctx}} \qquad \frac{\Gamma\,\text{ctx} \qquad A\,\text{type} \qquad x \notin \text{dom}(\Gamma)}{\Gamma, x : A\,\text{ctx}}$$

$$\frac{}{\Gamma, x : A, \Gamma' \vdash x : A} \qquad \frac{f \in \Sigma_2(A_0, \ldots; B) \qquad \Delta \vdash t_0 : A_0 \ \cdots}{\Delta \vdash f(t_0, \ldots) : B}$$

Figure 2: Preorder Type Theory: Type and Term Structure

respect to a context and an output type, contexts are ordered lists (this is important for our definition of dynamism context below). For bare preorder type theory, the only types are base types, and the only terms are variables and applications of uninterpreted function symbols . These are all given by a *signature* $\Sigma = (\Sigma_0, \Sigma_1, \Sigma_2, \Sigma_3)$, formally defined below in Definition 2.3. A substitution $\gamma : \Delta \vdash \Gamma$ is defined as usual:

Definition 2.1. A substitution $\gamma : \Delta \vdash \Gamma$ is a function which given a variable in the output context $x : A \in \Gamma$, produces a term of that type relative to the input context $\Delta \vdash \gamma(x) : A$.

Our term language supports a notion of substitution where if $\gamma : \Delta \vdash \Gamma$ and $\Gamma \vdash t : A$ then $\Delta \vdash t[\gamma] : A$, defined in the standard way for each construction we add. Weakening, contraction and exchange are all special cases of the admissible action of substitution.

Next, we discuss the new judgments of type dynamism, dynamism contexts, and term dynamism, shown in Figure 3. A type dynamism judgment $A \sqsubseteq B$ relates two well-formed types, and is read as "A is less dynamic than B". In preorder type theory, the only rules are reflexivity (TyDyn-Refl) and transitivity (TyDyn-Trans), which make type dynamism a preorder, and any axioms from the signature Σ_1 (TyDyn-Ax).

The remaining rules in Figure 3 define *type dynamism contexts* Φ, which are used in the definition of term dynamism. While terms are indexed by a type and a typing context, term dynamism judgments $\Phi \vdash t \sqsubseteq t' : A \sqsubseteq A'$ are indexed by two terms $\Gamma \vdash t : A$ and $\Gamma' \vdash t' : A'$, such that $A \sqsubseteq A'$ (A is less dynamic than A') and Γ is less dynamic than Γ'. Thus, we require a judgment $\Phi : \Gamma \sqsubseteq \Gamma'$, which lifts type dynamism to contexts pointwise (for any $x : A \in \Gamma$, the corresponding $x' : A' \in \Gamma'$ satisfies $A \sqsubseteq A'$). This uses the structure of Γ and Γ' as ordered lists: a dynamism context $\Phi : \Gamma \sqsubseteq \Gamma'$ implies that Γ and Γ' have the same length and associates variables based on their order in the context, so that Φ is uniquely determined by Γ and Γ'; if we want to form a judgment $t \sqsubseteq t'$ where their contexts are not aligned in this way, we can always use exchange on one of them to align it with the other. We notate dynamism contexts to evoke a logical relations interpretation of term dynamism: under the conditions that $x_0 \sqsubseteq x_0' : A_0 \sqsubseteq A_0', \ldots$ then we have that $t \sqsubseteq t' : B \sqsubseteq B'$.

The term dynamism judgment admits constructions (Figure 4) corresponding to both the structural rules of terms and the preorder structure of type dynamism, beginning from

$$\frac{}{A \sqsubseteq A}\ \textsc{TyDyn-Refl} \qquad \frac{A \sqsubseteq A' \qquad A' \sqsubseteq A''}{A \sqsubseteq A''}\ \textsc{TyDyn-Trans} \qquad \frac{(A, B) \in \Sigma_1}{A \sqsubseteq B}\ \textsc{TyDyn-Ax}$$

$$\frac{}{\cdot : \cdot \sqsubseteq \cdot} \qquad \frac{\Phi : \Gamma \sqsubseteq \Gamma' \qquad A \sqsubseteq A'}{(\Phi, x \sqsubseteq x' : A \sqsubseteq A') : \Gamma, x : A \sqsubseteq \Gamma', x' : A'}$$

Figure 3: Type and Context Dynamism

arbitrary term dynamism axioms (TmDyn-Ax). First, there is a rule (TmDyn-Var) that relates variables. Next there is a *compositionality* rule (TmDyn-Comp) that allows us to prove dynamism judgments by breaking terms down into components. This uses a notion of substitution dynamism $\Phi \vdash \gamma \sqsubseteq \gamma' : \Psi$ which is the pointwise extension of term dynamism to substitutions:

Definition 2.2. Given $\Phi : \Gamma \sqsubseteq \Gamma'$, $\Psi : \Delta \sqsubseteq \Delta'$, $\gamma : \Delta \vdash \Gamma$ and $\gamma' : \Delta' \vdash \Gamma'$, then

$$\Psi \vdash \gamma \sqsubseteq \gamma' : \Phi$$

is defined to hold when for every $x \sqsubseteq x' : A \sqsubseteq A' \in \Phi$, $\Psi \vdash \gamma(x) \sqsubseteq \gamma'(x') : A \sqsubseteq A'$

Last, we add an appropriate form of reflexivity (TmDyn-Refl) and transitivity (TmDyn-Trans) as rules, whose well-formedness depends on the reflexivity and transitivity of type dynamism. While the reflexivity rule is intuitive, the transitivity rule is more complex. Consider an example where $A \sqsubseteq A' \sqsubseteq A''$ and $B \sqsubseteq B' \sqsubseteq B''$:

$$\frac{x \sqsubseteq x' : A \sqsubseteq A' \vdash t \sqsubseteq t' : B \sqsubseteq B' \qquad x' \sqsubseteq x'' : A' \sqsubseteq A'' \vdash t' \sqsubseteq t'' : B' \sqsubseteq B''}{x \sqsubseteq x'' : A \sqsubseteq A'' \vdash t \sqsubseteq t'' : B \sqsubseteq B''}$$

In a logical relations interpretation of term dynamism, we would have relations $\sqsubseteq_{A,A'}$, $\sqsubseteq_{A',A''}$, $\sqsubseteq_{A,A''}$ and similarly for the B's, and the term dynamism judgment of the conclusion would be interpreted as saying that for any $u \sqsubseteq_{A,A''} u''$, $t[u/x] \sqsubseteq_{B,B''} t''[u''/x'']$. However, we could only instantiate the premises of the judgment if we could produce some middle u' with $u \sqsubseteq_{A,A'} u' \sqsubseteq_{A',A''} u''$. In such models, a middle u' *always* exists, because an implicit condition of the transitivity rule is that $\sqsubseteq_{A,A''}$ is the relation composite of $\sqsubseteq_{A,A'}$ and $\sqsubseteq_{A',A''}$ (the composite exists by type dynamism transitivity, and type dynamism witnesses are unique in PTT (thin in the semantics)). PTT itself does not give a term for this u', but the upcasts and downcasts in gradual type theory do (take it to be $\langle A' \nwarrow A \rangle u$ or $\langle A' \swarrow A'' \rangle u''$).

We also introduce some convenient syntactic sugar for term dynamism contexts and term dynamism, but for maximum clarity we will not use the sugar when introducing rules, only when it shortens proofs we present in the theory. Sometimes it is convenient to use the same variable name at the same type in both t and t' and so in such a case we simply write $x : A$, which, in a type dynamism context is just a macro for $x \sqsubseteq x : A \sqsubseteq A$ using the reflexivity of type dynamism. Then with this sugar, type contexts are a subset of type dynamism contexts. Similarly when t and t' have the same output type we write $\Phi \vdash t \sqsubseteq t' : A$ rather than the tediously long $\Phi \vdash t \sqsubseteq t' : A \sqsubseteq A$.

2.2. **PTT Signatures.** While gradual type theory proves that most operational rules of gradual typing are equivalences, some must be added as axioms. Compare Moggi's monadic metalanguage [22]: since it is a general theory of monads, it is not provable that an effect is

$$\frac{x \sqsubseteq x' : A \sqsubseteq A' \in \Phi}{\Phi \vdash x \sqsubseteq x' : A \sqsubseteq A'}\ \textsc{TmDyn-Var} \qquad \frac{\Phi \vdash t \sqsubseteq t' : A \sqsubseteq A' \qquad \Psi \vdash \gamma \sqsubseteq \gamma' : \Phi}{\Psi \vdash t[\gamma] \sqsubseteq t'[\gamma'] : A \sqsubseteq A'}\ \textsc{TmDyn-Comp}$$

$$\frac{\Gamma \vdash t : A \qquad \Phi : \Gamma \sqsubseteq \Gamma}{\Phi \vdash t \sqsubseteq t : A \sqsubseteq A}\ \textsc{TmDyn-Refl} \qquad \frac{\begin{array}{c}\Phi : \Gamma \sqsubseteq \Gamma' \vdash t \sqsubseteq t' : A \sqsubseteq A' \\ \Phi' : \Gamma' \sqsubseteq \Gamma'' \vdash t' \sqsubseteq t'' : A' \sqsubseteq A'' \\ \Psi : \Gamma \sqsubseteq \Gamma''\end{array}}{\Psi : \Gamma \sqsubseteq \Gamma'' \vdash t \sqsubseteq t'' : A \sqsubseteq A''}\ \textsc{TmDyn-Trans}$$

$$\frac{(t, t') \in \Sigma_3 \qquad \Gamma \vdash t : A \qquad \Gamma' \vdash t' : A' \qquad \Phi : \Gamma \sqsubseteq \Gamma'}{\Phi \vdash t \sqsubseteq t' : A \sqsubseteq A'}\ \textsc{TmDyn-Ax}$$

Figure 4: Primitive Rules of Term Dynamism

commutative, but we can add a commutativity axiom and prove additional consequences. Similarly, in our type theory it is not provable without adding additional axioms that an upcast followed by its complementary downcast is the identity, or that the function type and product type are disjoint. To allow such axioms, preorder type theory is formally a *family* of type theories parameterized by a *signature*; the signature is also needed for a precise categorical semantics, because it represents the "generating data" of a specific model.

The signatures for preorder type theory (and, below, gradual type theory) package together all of the base types, uninterpreted function symbols and type and term dynamism axioms we desire. This is mutually defined with the definition of the type theory itself, so that for instance we can add function symbols whose codomain is a non-base type.

Definition 2.3 (PTT Signature)**.** The notion of preorder type theory signature (PTT signature) is built as follows

(1) A 0-PTT signature is a set, and elements are called *base types*.
(2) For a 0-PTT signature Σ_0, $PTT_0(\Sigma_0)$ is the set of types generated by that signature and the rules of preorder type theory.
(3) A 1-PTT Signature relative to a 0-PTT signature Σ_0 is a subset of $PTT_0(\Sigma_0)^2$, and elements are called *type dynamism axioms*.
(4) A 2-PTT Signature relative to a 0-PTT signature Σ_0 is a set Σ_2 with functions $s : \Sigma_2 \to PTT_0(\Sigma_0)^*$ and $t : \Sigma_2 \to \mathrm{PTT}_0(\Sigma_0)$, and whose elements are called *function symbols*.[2] We define $\Sigma_2(A_0, \ldots; B) = \{t \in \Sigma_2 \mid s(t) = A_0, \ldots \wedge t(t) = B\}$
(5) For $0, 1, 2$-PTT signatures $\Sigma_0, \Sigma_1, \Sigma_2$, define $PTT_1(\Sigma_0, \Sigma_1, \Sigma_2)$ to be the set of all terms in PTT generated from the signatures.
(6) A 3-PTT Signature Σ_3 relative to $0, 1, 2$-signatures $\Sigma_0, \Sigma_1, \Sigma_2$ is a set
$$\Sigma_3 \subseteq PTT_1(\Sigma_0, \Sigma_1, \Sigma_2)^2$$
such that if $(t, t') \in \Sigma_3$ and $\Gamma \vdash t : A$ and $\Gamma' \vdash t' : A'$, then it is derivable using $\Sigma_0, \Sigma_1, \Sigma_2$ that $\Gamma \sqsubseteq \Gamma'$ and $A \sqsubseteq A'$. Elements of Σ_3 are called *term dynamism axioms*.
(7) Finally a PTT signature is a tuple of $0, 1, 2, 3$-PTT signatures $(\Sigma_0, \Sigma_1, \Sigma_2, \Sigma_3)$, each relative to the previous signatures.

[2] technically the dependency on Σ_1 is trivial here, but is needed when we extend to GTT signatures.

2.3. **Gradual Type Theory.** Preorder Type Theory gives us a simple foundation with which to build Gradual Type Theory in a modular way: we can characterize different aspects of gradual typing, such as a dynamic type, casts, and type errors separately.

2.3.1. *Casts.* We start by defining upcasts and downcasts, using type and term dynamism in Figure 5. Given that $A \sqsubseteq A'$, the upcast is a function from A to A' such that for any $t : A$, $\langle A' \nwarrow A\rangle t$ is the *least dynamic term of type A' that is at least as dynamic as t*. The UR rule can be thought of as the "introduction rule", saying $\langle A' \nwarrow A\rangle x$ is more dynamic than x, and then UL is the "elimination rule", saying that if some $x' : A'$ is more dynamic than $x : A$, then it is more dynamic than $\langle A' \nwarrow A\rangle x$ — since $\langle A' \nwarrow A\rangle x$ is the *least* dynamic term with this property. The rules for projections are dual, ensuring that for $x' : A'$, $\langle A \swarrow A'\rangle x'$ is the most dynamic term of type A that is less dynamic than x'.

In fact, combined with the TmDyn-Trans rule, we can show that it has a slightly more general property: $\langle A' \nwarrow A\rangle x$ is not just less dynamic than any term of type A' more dynamic than x, but is less dynamic than any term of type A' *or higher*, i.e. of type $A'' \sqsupseteq A'$. Indeed, it is often convenient to use the following sequent-calculus style rules (everything in the conclusion is fully general, except for one cast), which are derivable using the TmDyn-Trans and TmDyn-Comp

$$\frac{\Phi \vdash t \sqsubseteq t' : A \sqsubseteq A' \qquad A' \sqsubseteq A''}{\Phi \vdash t \sqsubseteq \langle A'' \nwarrow A'\rangle t' : A \sqsubseteq A''}\ \mathrm{UR(S)} \qquad \frac{\Phi \vdash t \sqsubseteq t'' : A \sqsubseteq A'' \quad A \sqsubseteq A' \qquad A' \sqsubseteq A''}{\Phi \vdash \langle A' \nwarrow A\rangle t \sqsubseteq t'' : A' \sqsubseteq A''}\ \mathrm{UL(S)}$$

$$\frac{\Phi \vdash t' \sqsubseteq t'' : A' \sqsubseteq A'' \qquad A \sqsubseteq A'}{\Phi \vdash \langle A \swarrow A'\rangle t' \sqsubseteq t'' : A \sqsubseteq A''}\ \mathrm{DL(S)} \qquad \frac{\Phi \vdash t \sqsubseteq t'' : A \sqsubseteq A'' \quad A \sqsubseteq A' \qquad A' \sqsubseteq A''}{\Phi \vdash t \sqsubseteq \langle A' \swarrow A''\rangle t'' : A \sqsubseteq A'}\ \mathrm{DR(S)}$$

In particular, the upcast is left-invertible, and the downcast is right-invertible (which agrees with their status as left and right adjoints discussed below).

Though when read "top-down" the UL(S) and DR(S) rules have more side-conditions in them than the UR(S),DL(S) rules, when read "bottom-up", they require fewer assumptions. That is, in UL(S) (and similarly DR(S)) if we know the conclusion is *well-formed*, as we would when constructing a proof, then the assumption that $A \sqsubseteq A'$ follows from the fact that the upcast $\langle A' \nwarrow A\rangle t$ is well-formed, $A' \sqsubseteq A''$ follows from the typing, and finally $A \sqsubseteq A''$ follows by transitivity. On the other hand, in UR(S) (similarly DL(S)), for the premise to be well-formed, we need to know that $A \sqsubseteq A'$ which does *not* follow from the well-formedness of the conclusion. So from a proof-construction standpoint, we can always apply a rule when we have an upcast on the left or a downcast on the right, but we must check a side-condition when we have an upcast on the right or downcast on the left.

As we will discuss in Section 3, these rules allow us to prove that the pair of the upcast and downcast form a *Galois connection* (adjunction), meaning $\langle A' \nwarrow A\rangle\langle A \swarrow A'\rangle t \sqsubseteq t$ and $t \sqsubseteq \langle A \swarrow A'\rangle\langle A' \nwarrow A\rangle t$. However in existing gradually typed languages, the casts satisfy the stronger condition of being a *Galois insertion*, in which the left adjoint, the downcast, is a *retract* of the upcast, meaning $t \sqsupseteq\sqsubseteq \langle A \swarrow A'\rangle\langle A' \nwarrow A\rangle t$. We can restrict to Galois insertions by adding the *retract axiom* Retract. Most theorems of gradual type theory do not require it, though this axiom is satisfied in all models of preorder type theory in Section 6.

$$\frac{\Gamma \vdash t : A \qquad A \sqsubseteq A'}{\Gamma \vdash \langle A' \nwarrow A \rangle t : A'} \qquad \frac{\Gamma \vdash t : A' \qquad A \sqsubseteq A'}{\Gamma \vdash \langle A \swarrow A' \rangle t : A}$$

$$\frac{A \sqsubseteq A'}{x \sqsubseteq x : A \sqsubseteq A \vdash x \sqsubseteq \langle A' \nwarrow A \rangle x : A \sqsubseteq A'}\ \text{UR}$$

$$\frac{A \sqsubseteq A'}{x' \sqsubseteq x' : A' \sqsubseteq A' \vdash \langle A \swarrow A' \rangle x' \sqsubseteq x' : A \sqsubseteq A'}\ \text{DL}$$

$$\frac{A \sqsubseteq A'}{x \sqsubseteq x' : A \sqsubseteq A' \vdash \langle A' \nwarrow A \rangle x \sqsubseteq x' : A' \sqsubseteq A'}\ \text{UL}$$

$$\frac{A \sqsubseteq A'}{x \sqsubseteq x' : A \sqsubseteq A' \vdash x \sqsubseteq \langle A \swarrow A' \rangle x' : A \sqsubseteq A}\ \text{DR}$$

$$\frac{A \sqsubseteq A'}{x : A \sqsubseteq x : A \vdash \langle A \swarrow A' \rangle \langle A' \nwarrow A \rangle x \sqsubseteq x : A}\ \textsc{Retract}$$

$$\frac{}{? \,\text{type}} \qquad \frac{}{A \sqsubseteq ?}\ ?\textsc{Top} \qquad \frac{}{\Gamma \vdash \mho_A : A} \qquad \frac{\Phi : \Gamma \sqsubseteq \Gamma}{\Phi \vdash \mho_A \sqsubseteq t : A}\ \mho\textsc{Bot}$$

Figure 5: Upcasts, Downcasts, Dynamic Type and Type Error

2.3.2. *Dynamic Type and Type Errors.* The remaining rules in Figure 5 define the dynamic type and type errors, which are also given a universal property in terms of type and term dynamism. The dynamic type is defined as the most dynamic type (?Top). The type error, written as $\mho$, is defined by the fact that it is a constant at every type A that is a least element of A ($\mho$Bot). By transitivity, this further implies that $\mho_A \sqsubseteq t : A \sqsubseteq A'$ for any $A' \sqsupseteq A$.

2.3.3. *Negative Connectives.* Next we illustrate how simple negative types can be defined in preorder type theory in Figure 6.

The first portion of the figure presents the rules for function types. First, we have a rule to say a function type is well-formed, and $\to$Mon states that function types are monotone in *both* arguments with respect to term dynamism, following previous work on type dynamism [14, 42, 33]. Because of this, type dynamism is sometimes referred to as "naïve subtyping". See Section 5.1 for a semantic explanation of the meaning of $\to$Mon. Next we have standard typing rules for λ and application, and corresponding monotonicity rules λMon and AppMon for term dynamism. Finally, we have call-by-name β and η rules, which we present as equi-dynamism: we write $\sqsupseteq\sqsubseteq$ to mean a rule exists in each direction.

Next, the presentation of product types is much the same: well-formedness and monotonicity of the type constructor, standard introduction and elimination rules with corresponding monotonicity rules and finally call-by-name $\beta\eta$ rules. Lastly, we have three rules for the unit type. First, we have a well-formedness rule, the corresponding monotonicity

rule is unnecessary because it follows from reflexivity of type dynamism (TyDyn-Refl). Next, we have introduction, whose monotonicity follows from reflexivity of term dynamism (TmDyn-Refl). Finally we include the unit type's η law. There is no β law because there is no elimination rule.

Specifically, we present the unit type, products and function types in Figure 6. The type and term constructors are the same as those in the simply typed λ-calculus. Each type constructor extends type dynamism in the standard way : every connective is *monotone* in every argument, including the function type. Due to the covariance of the function type, For term dynamism, we add two classes of rules. First, there are congruence rules that "extrude" the term constructor rules for the type, which are like a "congruence of contextual approximation" condition. Next, the computational rules reflect the ordinary β, η equivalences as equi-dynamism:

We call the accumulation of all of these connectives *gradual type theory*. A gradual type theory signature is a PTT signature where each declaration can additionally use the structure of gradual type theory:

Definition 2.4 (GTT Signature)**.** A GTT signature $(\Sigma_0, \Sigma_1, \Sigma_2, \Sigma_3)$ is a PTT signature, where each declaration may make use of the rules for dynamic type, casts, type error, functions, products and unit types, in addition to the rules of PTT.

3. Theorems and Constructions in Gradual Type Theory

In this section, we discuss some of the consequences of the axioms of gradual type theory in the form of *theorems* derivable in gradual type theory. These theorems come in the form of term orderings and equivalences. If we accept that gradual type theory is a reasonable axiomatization of a gradually typed language satisfying $\beta\eta$ equivalence and graduality, then the equivalences we derive here imply program equivalences in any such language. We divide the theorems we show into two groups. First, we present the "reductions", i.e., equivalences that correspond to operational reductions in gradually typed languages. Second, we present theorems that are not operational reductions, but more abstract properties of casts, that help us relate to the denotational semantics and other presentations of graduality.

3.1. Cast Reduction Theorems.

We now present several theorems that are typically the part of the operational semantics of a gradually typed language. Since we are able to *derive* these as theorems, this shows that they are essential components of a language satisfying $\beta\eta$ and graduality. For instance, if any of these program equivalences is violated, then a language must violate β, η or graduality.

First we show that the upcast and downcast from a type to itself are the identity function.

Theorem 3.1 (Identity Casts)**.** $\langle A \nwarrow A\rangle t \sqsupseteq\sqsubseteq t$ *and* $\langle A \swarrow A\rangle t \sqsupseteq\sqsubseteq t$.

Proof. The intuition is simple: given $t : A$, t *itself* is the least dynamic element of A that is at least as dynamic as t. For a formal proof, we show that $x : A$ and $\langle A \nwarrow A\rangle x$ are equi-dynamic and each direction is an instance of UL or UR:

$$\frac{}{x : A \vdash x \sqsubseteq \langle A \nwarrow A\rangle x : A}\ \text{UR} \qquad \frac{}{x : A \sqsubseteq x : A \vdash \langle A \nwarrow A\rangle x \sqsubseteq x : A \sqsubseteq A}\ \text{UL}$$

$$\frac{A\,\mathrm{type} \qquad B\,\mathrm{type}}{A \to B\,\mathrm{type}} \qquad \frac{A \sqsubseteq A' \qquad B \sqsubseteq B'}{A \to B \sqsubseteq A' \to B'}\ {\to}\textsc{Mon}$$

$$\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x : A.t : A \to B} \qquad \frac{\Gamma \vdash t : A \to B \qquad \Gamma \vdash u : A}{\Gamma \vdash t\,u : B}$$

$$\frac{\Phi, x \sqsubseteq x' : A \sqsubseteq A' \vdash t \sqsubseteq t' : B \sqsubseteq B'}{\Phi \vdash \lambda x : A.t \sqsubseteq \lambda x' : A'.t' : A \to B \sqsubseteq A' \to B'}\ \lambda\textsc{Mon}$$

$$\frac{\begin{array}{c}\Phi \vdash t \sqsubseteq t' : A \to B \sqsubseteq A' \to B' \\ \Phi \vdash u \sqsubseteq u' : A \sqsubseteq A'\end{array}}{\Phi \vdash t\,u \sqsubseteq t'\,u' : B \sqsubseteq B'}\ \textsc{AppMon}$$

$$\frac{\begin{array}{c}\Phi : \Gamma \sqsubseteq \Gamma \\ \Gamma \vdash t : A \to B \qquad \Gamma \vdash u : A\end{array}}{\Phi \vdash (\lambda x : A.t)u \sqsupseteq\sqsubseteq t[u/x] : B \sqsubseteq B}\ {\to}\beta \qquad \frac{\begin{array}{c}\Phi : \Gamma \sqsubseteq \Gamma \\ \Gamma \vdash t : A \to B\end{array}}{\Phi \vdash t \sqsupseteq\sqsubseteq (\lambda x : A.t\,x) : A \to B \sqsubseteq A \to B}\ {\to}\eta$$

$$\frac{A_1\,\mathrm{type} \qquad A_2\,\mathrm{type}}{A_1 \times A_2\,\mathrm{type}} \qquad \frac{A_1 \sqsubseteq A_1' \qquad A_2 \sqsubseteq A_2'}{A_1 \times A_2 \sqsubseteq A_1' \times A_2'}\ {\times}\textsc{Mon}$$

$$\frac{\Gamma \vdash t_1 : A_1 \qquad \Gamma \vdash t_2 : A_2}{\Gamma \vdash (t_1, t_2) : A_1 \times A_2} \qquad \frac{\Gamma \vdash t : A_1 \times A_2 \qquad i \in 1, 2}{\Gamma \vdash \pi_i t : A_i}$$

$$\frac{\Phi \vdash t_1 \sqsubseteq t_1' : A_1 \sqsubseteq A_1' \qquad \Phi \vdash t_2 \sqsubseteq t_2' : A_2 \sqsubseteq A_2'}{\Phi \vdash (t_1, t_2) \sqsubseteq (t_1', t_2') : A_1 \times A_2 \sqsubseteq A_1' \times A_2'}\ \textsc{PairMon}$$

$$\frac{\Phi \vdash t \sqsubseteq t' : A_1 \times A_2 \sqsubseteq A_1' \times A_2'}{\Phi \vdash \pi_i t \sqsubseteq \pi_i t' : A_i \sqsubseteq A_i'}\ \textsc{PrjMon}$$

$$\frac{i \in \{1, 2\}}{\Gamma \vdash \pi_i(t_1, t_2) \sqsupseteq\sqsubseteq t_i : A_i}\ {\times}\beta \qquad \frac{}{\Gamma \vdash t \sqsupseteq\sqsubseteq (\pi_1 t, \pi_2 t) : A_1 \times A_2}\ {\times}\eta$$

$$\frac{}{1\,\mathrm{type}} \qquad \frac{}{\Gamma \vdash () : 1} \qquad \frac{}{\Gamma \vdash t \sqsupseteq\sqsubseteq () : 1}\ 1\eta$$

Figure 6: Function, Product and Unit Types

The downcast has a perfectly dual proof. □

Since this is our first example of using the cast term dynamism rules, it is instructive to note that, given $A \sqsubseteq A'$ so that $\langle A' \leftharpoondown A \rangle$ is well-defined, we *cannot* show that

$\langle A' \nwarrow A \rangle x \sqsubseteq x$ analogously to the second derivation

$$\frac{}{x : A \sqsubseteq y : A \vdash \langle A' \nwarrow A \rangle x \sqsubseteq y : A' \sqsubseteq A} \text{ NOT AN INSTANCE OF UL}$$

because the conclusion violates the presupposition of the judgment, which would require $A' \sqsubseteq A$, and is moreover not an instance of UL, which would require y to have type A', not type A. That is, the existence of appropriate type dynamism relations is crucial to these rules, so it is important to be careful about the types involved.

Next, we show that if $A \sqsubseteq A' \sqsubseteq A''$, then the upcast from A to A'' factors through A', and dually for the downcast from A'' to A. This justifies the operational rule familiar in gradual typing that separates the function contract into the "higher-order" part that proxies the original function and the "first-order" tag checking:

$$\langle ? \nwarrow A \to B \rangle t \mapsto \langle ? \nwarrow ? \to ? \rangle \langle ? \to ? \nwarrow A \to B \rangle t$$

More generally, it implies that casts from A to A' where $A \sqsubseteq A'$ commute over the dynamic type, e.g. $\langle ? \nwarrow A' \rangle \langle A' \nwarrow A \rangle x \sqsupseteq\sqsubseteq \langle ? \nwarrow A \rangle x$—intuitively, if casts only perform checks, and do not change values, then a value's representation in the dynamic type should not depend on how it got there. This can also justify some *optimizations* of gradual programs, collapsing multiple casts into one. This property, combined with the identity property, also says that upcasts and downcasts form respective *subcategories* of arbitrary terms (the composition of two upcasts (downcasts) is an upcast (downcast) and identity terms are also upcasts and downcasts), and that the upcasts and downcasts each determine functors from the category of types and type dynamism relations to the category of types and terms.

Theorem 3.2 (Casts (De-)Compose)**.** *If $A \sqsubseteq A' \sqsubseteq A''$, then $\langle A'' \nwarrow A \rangle t \sqsupseteq\sqsubseteq \langle A'' \nwarrow A' \rangle \langle A' \nwarrow A \rangle t$ and dually, $\langle A \swarrow A'' \rangle t \sqsupseteq\sqsubseteq \langle A \swarrow A' \rangle \langle A' \swarrow A'' \rangle t$.*

Proof. The proofs are dual, so we only show the argument for upcasts. We want to show $\langle A'' \nwarrow A \rangle x \sqsupseteq\sqsubseteq \langle A'' \nwarrow A' \rangle \langle A' \nwarrow A \rangle x$. On the one hand, to show something of type A'' is more dynamic than $\langle A'' \nwarrow A \rangle x$, we just have to show that it is more dynamic than x, which is true of $\langle A'' \nwarrow A' \rangle \langle A' \nwarrow A \rangle x$. The other direction is similar, first we peel off $\langle A'' \nwarrow A' \rangle \cdot$ and then $\langle A' \nwarrow A \rangle \cdot$. More formally, assuming $A \sqsubseteq A' \sqsubseteq A''$, the following are valid derivations:

$$\dfrac{\dfrac{\dfrac{}{x : A \vdash x \sqsubseteq \langle A' \nwarrow A \rangle x : A \sqsubseteq A'} \text{ UR}}{x : A \vdash x \sqsubseteq \langle A'' \nwarrow A' \rangle \langle A' \nwarrow A \rangle x : A \sqsubseteq A''} \text{ UR(S)}}{x : A \vdash \langle A'' \nwarrow A \rangle x \sqsubseteq \langle A'' \nwarrow A' \rangle \langle A' \nwarrow A \rangle x : A''} \text{ UL(S)}$$

$$\dfrac{\dfrac{\dfrac{}{x : A \vdash x \sqsubseteq \langle A'' \nwarrow A \rangle x : A \sqsubseteq A''} \text{ UR}}{x : A \vdash \langle A' \nwarrow A \rangle x \sqsubseteq \langle A'' \nwarrow A \rangle x : A' \sqsubseteq A''} \text{ UL(S)}}{x : A \vdash \langle A'' \nwarrow A' \rangle \langle A' \nwarrow A \rangle x \sqsubseteq \langle A'' \nwarrow A \rangle x : A''} \text{ UL(S)}$$

□

Next, we show the most important theorems, which state that casts between function types must be implemented by the functorial action of the function type on casts (and the analogous case for products). This reproduces the standard "wrapping" implementation of

function and product casts [11, 10]. Each proof is modular (depends on no more type or term constructors other than those related to function and product types respectively).

Theorem 3.3 (Function and Product Cast Reductions)**.** *Whenever $A \sqsubseteq A'$ and $B \sqsubseteq B'$, the following are all satisfied.*

(1) $\langle A' \to B' \leftharpoondown A \to B\rangle t \sqsupseteq\sqsubseteq \lambda x : A'.\langle B' \leftharpoondown B\rangle(t(\langle A \upharpoonleft A'\rangle x))$
(2) $\langle A \to B \upharpoonleft A' \to B'\rangle t \sqsupseteq\sqsubseteq \lambda x : A.\langle B \upharpoonleft B'\rangle(t(\langle A' \leftharpoondown A\rangle x))$.
(3) $\langle A'_0 \times A'_1 \leftharpoondown A_0 \times A_1\rangle t \sqsupseteq\sqsubseteq (\langle A'_0 \leftharpoondown A_0\rangle \pi_0 t, \langle A'_1 \leftharpoondown A_1\rangle \pi_1 t)$
(4) $\langle A_0 \times A_1 \upharpoonleft A'_0 \times A'_1\rangle t \sqsupseteq\sqsubseteq (\langle A_0 \upharpoonleft A'_0\rangle \pi_0 t, \langle A_1 \upharpoonleft A'_1\rangle \pi_1 t)$.

Proof. To make the function proof easier to understand, we first derive a higher-level "extensionality principle", which the reader may find to be more intuitive than the η principle: a function is less dynamic than another if applying it to a less dynamic input yields a less dynamic result:

$$\frac{\Phi, x \sqsubseteq x' : A \sqsubseteq A' \vdash tx \sqsubseteq t'x' : B \sqsubseteq B'}{\Phi \vdash t \sqsubseteq t' : A \to B \sqsubseteq A' \to B'}\ \textsc{fun-ext}$$

It follows from the η principles and the congruence rules for λ:

$$\frac{t \sqsubseteq \lambda x.tx \qquad \lambda x'.t'x' \sqsubseteq t' \qquad \dfrac{\Phi, x \sqsubseteq x' : A \sqsubseteq A' \vdash tx \sqsubseteq t'x' : B \sqsubseteq B'}{\Phi \vdash \lambda x.tx \sqsubseteq \lambda x'.t'x' : A \to B \sqsubseteq A' \to B'}}{\Phi \vdash t \sqsubseteq t' : A \to B \sqsubseteq A' \to B'}\ \textsc{tmprec-trans}$$

For the function contract, we need to show

$$\langle A' \to B' \leftharpoondown A \to B\rangle f \sqsupseteq\sqsubseteq \lambda x' : A'.\langle B' \leftharpoondown B\rangle(f(\langle A \upharpoonleft A'\rangle x')).$$

First to show $\sqsubseteq$, it is sufficient to show that the right hand side is more dynamic than f itself. Next we invoke the extensionality principle (FUN-EXT) and β and then we have to show that $x \sqsubseteq x' : A \sqsubseteq A' \vdash fx \sqsubseteq \langle B' \leftharpoondown B\rangle(f(\langle A \upharpoonleft A'\rangle x'))$. This follows from congruence of application and the rules of casts. As a derivation tree:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{f, x \sqsubseteq x' \vdash x \sqsubseteq x'}{f, x \sqsubseteq x' \vdash x \sqsubseteq \langle A \upharpoonleft A'\rangle x'} \qquad f, x \sqsubseteq x' \vdash f \sqsubseteq f}{f, x \sqsubseteq x' \vdash fx \sqsubseteq f(\langle A \upharpoonleft A'\rangle x')}}{f, x \sqsubseteq x' \vdash fx \sqsubseteq \langle B' \leftharpoondown B\rangle(f(\langle A \upharpoonleft A'\rangle x'))}}{f, x \sqsubseteq x' : A \sqsubseteq A' \vdash fx \sqsubseteq (\lambda x' : A'.\langle B' \leftharpoondown B\rangle(f(\langle A \upharpoonleft A'\rangle x')))x'}}{f \vdash f \sqsubseteq \lambda x' : A'.\langle B' \leftharpoondown B\rangle(f(\langle A \upharpoonleft A'\rangle x'))}}{f : A \to B \vdash \langle A' \to B' \leftharpoondown A \to B\rangle f \sqsubseteq \lambda x' : A'.\langle B' \leftharpoondown B\rangle(f(\langle A \upharpoonleft A'\rangle x'))}$$

For the opposite direction, we invoke the extensionality principle and β reduce, then needing to show $\langle B' \leftharpoondown B\rangle(f(\langle A \upharpoonleft A'\rangle x')) \sqsubseteq (\langle A' \to B' \leftharpoondown A \to B\rangle f)x'$. We can remove

the upcast from the left and then use the congruence rules. As a derivation tree:

$$\dfrac{\dfrac{\dfrac{\dfrac{f, x' \vdash f \sqsubseteq f}{f, x' \vdash f \sqsubseteq \langle A' \to B' \nwarrow A \to B\rangle f} \quad \dfrac{f, x' \vdash x' \sqsubseteq x'}{f, x' \vdash \langle A \swarrow A'\rangle x' \sqsubseteq x'}}{f, x' \vdash f(\langle A \swarrow A'\rangle x') \sqsubseteq (\langle A' \to B' \nwarrow A \to B\rangle f)x'}}{f, x' \vdash \langle B' \nwarrow B\rangle(f(\langle A \swarrow A'\rangle x')) \sqsubseteq (\langle A' \to B' \nwarrow A \to B\rangle f)x'}}{f, x' : A' \vdash (\lambda x' : A'.\langle B' \nwarrow B\rangle(f(\langle A \swarrow A'\rangle x')))x' \sqsubseteq (\langle A' \to B' \nwarrow A \to B\rangle f)x'}}{f : A \to B \vdash \lambda x' : A'.\langle B' \nwarrow B\rangle(f(\langle A \swarrow A'\rangle x')) \sqsubseteq \langle A' \to B' \nwarrow A \to B\rangle f}$$

The downside of using the extensionality principle is that we need to use β reduction, when in actuality we only need to use η in the proof. In figure 7, we present "direct" proofs for function and product upcasts that use only η equivalence, and not the extensionality principle or β. The proofs for downcasts are exactly dual. □

We note that in the proofs above, each direction of the equivalence depends only on *one* direction of η equivalence, and so in gradual languages where η only holds as an ordering, we still get an ordering relationship with the wrapping semantics.

Finally, we can show that upcasts must preserve errors, and if the retract axiom is assumed, downcasts must as well.

Theorem 3.4 (Casts are strict)**.** *If $A \sqsubseteq A'$, then*
(1) $\langle A' \nwarrow A\rangle \mathcal{U}_A \sqsupseteq\sqsubseteq \mathcal{U}_{A'}$
(2) *Assuming the retract axiom,* $\langle A \swarrow A'\rangle \mathcal{U}_A \sqsupseteq\sqsubseteq \mathcal{U}_{A'}$

Proof. The upcast preserves $\mathcal{U}$ because it is a left/upper adjoint and therefore preserves colimits/joins, and $\mathcal{U}$ is the empty join. More concretely, $\langle A' \nwarrow A\rangle \mathcal{U}_A \sqsubseteq \mathcal{U}_{A'}$ because $\mathcal{U}_{A'}$ is more dynamic than $\mathcal{U}_A$ and is the least dynamic term of type A' so is in particular less dynamic than anything more dynamic than $\mathcal{U}_A$. As derivations:

$$\dfrac{}{\cdot \vdash \mathcal{U}_{A'} \sqsubseteq \langle A' \nwarrow A\rangle \mathcal{U}_A : A'}\ \textsc{Err-Bot} \qquad \dfrac{\dfrac{}{\cdot \vdash \mathcal{U}_A \sqsubseteq \mathcal{U}_{A'} : A \sqsubseteq A'}\ \textsc{Err-Bot'}}{\cdot \vdash \langle A' \nwarrow A\rangle \mathcal{U}_A \sqsubseteq \mathcal{U}_{A'} : A'}\ \mathrm{UL(S)}$$

The proof that the downcast preserves $\mathcal{U}$ is less modular as it depends on the presence of the upcast and the retract axiom. The proof is simple though: to show $\langle A \swarrow A'\rangle \mathcal{U}_{A'} \sqsupseteq\sqsubseteq \mathcal{U}_A : A$, we have $\mathcal{U}_{A'} \sqsupseteq\sqsubseteq \langle A' \nwarrow A\rangle \mathcal{U}_A$ by above, and so we can apply the downcast to both sides to get $\langle A \swarrow A'\rangle \mathcal{U}_{A'} \sqsupseteq\sqsubseteq \langle A \swarrow A'\rangle\langle A' \nwarrow A\rangle \mathcal{U}_A$, and the right-hand side is equivalent to $\mathcal{U}_A$ by the retract axiom. □

3.2. **Properties of Casts.** Next we present a few properties of casts that are not operational reductions, but have other semantic significance.

First, we make precise the idea that the specification we have given for casts characterizes them uniquely up to order-equivalence ($\sqsupseteq\sqsubseteq$). In category theoretic terms, we show that the rules for upcasts/downcasts constitute a *universal property*. First, to prove that casts are unique, suppose that there was a second version of the upcast $\langle A' \nwarrow A\rangle t$ with analogous sequent-style rules UL(S)' and UR(S)'.

Theorem 3.5 (Upcasts are Unique)**.** *If $A \sqsubseteq A'$, then $\langle A' \nwarrow A\rangle x \sqsupseteq\sqsubseteq \langle A' \nwarrow A\rangle x$.*

$$\dfrac{f \sqsubseteq \lambda x.fx \qquad \dfrac{\dfrac{\dfrac{\dfrac{f, x \sqsubseteq x' \vdash f \sqsubseteq f : A \to B \sqsubseteq A \to B \qquad \dfrac{f, x \sqsubseteq x' \vdash x \sqsubseteq x' : A \sqsubseteq A'}{f, x \sqsubseteq x' \vdash x \sqsubseteq \langle A \nwarrow A' \rangle x' : A \sqsubseteq A}}{f, x \sqsubseteq x' \vdash fx \sqsubseteq f(\langle A \nwarrow A' \rangle x') : B \sqsubseteq B}}{f, x \sqsubseteq x' \vdash fx \sqsubseteq \langle B' \nwarrow B \rangle (f(\langle A \nwarrow A' \rangle x')) : B \sqsubseteq B'}}{\lambda x.fx \sqsubseteq \lambda x' : A'.\langle B' \nwarrow B \rangle (f(\langle A \nwarrow A' \rangle x'))}}{f \vdash f \sqsubseteq \lambda x' : A'.\langle B' \nwarrow B \rangle (f(\langle A \nwarrow A' \rangle x')) : A \to B \sqsubseteq A' \to B'}}{f : A \to B \vdash \langle A' \to B' \nwarrow A \to B \rangle f \sqsubseteq \lambda x' : A'.\langle B' \nwarrow B \rangle (f(\langle A \nwarrow A' \rangle x')) : A' \to B'}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{f \vdash f \sqsubseteq (\langle A' \to B' \nwarrow A \to B \rangle f) \qquad x' \vdash \langle A \nwarrow A' \rangle x' \sqsubseteq x'}{f, x' \vdash f(\langle A \nwarrow A' \rangle x') \sqsubseteq (\langle A' \to B' \nwarrow A \to B \rangle f) x'}}{f, x' : A' \vdash \langle B' \nwarrow B \rangle (f(\langle A \nwarrow A' \rangle x')) \sqsubseteq (\langle A' \to B' \nwarrow A \to B \rangle f) x'}}{f \vdash \lambda x' : A'.\langle B' \nwarrow B \rangle (f(\langle A \nwarrow A' \rangle x')) \sqsubseteq \lambda x'.(\langle A' \to B' \nwarrow A \to B \rangle f) x'}\ \mathcal{D}$$

$$\dfrac{\mathcal{D} \qquad \lambda x'.tx' \sqsubseteq t \text{ with } t = \langle A' \to B' \nwarrow A \to B \rangle f}{f : A \to B \vdash \lambda x' : A'.\langle B' \nwarrow B \rangle (f(\langle A \nwarrow A' \rangle x')) \sqsubseteq \langle A' \to B' \nwarrow A \to B \rangle f : A' \to B'}$$

$$\dfrac{p \sqsubseteq (\pi_0 p, \pi_1 p) \qquad \dfrac{\dfrac{\dfrac{\pi_i p \sqsubseteq \pi_i p}{\forall i \in 0, 1.\, \pi_i p \sqsubseteq \langle A_i' \nwarrow A_i \rangle \pi_i p}}{p \vdash (\pi_0 p, \pi_1 p) \sqsubseteq (\langle A_0' \nwarrow A_0 \rangle \pi_0 p, \langle A_1' \nwarrow A_1 \rangle \pi_0 p)}}{p \vdash p \sqsubseteq (\langle A_0' \nwarrow A_0 \rangle \pi_0 p, \langle A_1' \nwarrow A_1 \rangle \pi_0 p)}}{p : A_0 \times A_1 \vdash \langle A_0' \times A_1' \nwarrow A_0 \times A_1 \rangle p \sqsubseteq (\langle A_0' \nwarrow A_0 \rangle \pi_0 p, \langle A_1' \nwarrow A_1 \rangle \pi_0 p) : A_0' \times A_1'}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{p \sqsubseteq p}{p \sqsubseteq \langle A_0' \times A_1' \nwarrow A_0 \times A_1 \rangle p}}{\pi_i p \sqsubseteq \pi_i \langle A_0' \times A_1' \nwarrow A_0 \times A_1 \rangle p}}{\forall i \in 0, 1.\, \langle A_i' \nwarrow A_i \rangle \pi_i p \sqsubseteq \pi_i \langle A_0' \times A_1' \nwarrow A_0 \times A_1 \rangle p}}{(\langle A_0' \nwarrow A_0 \rangle \pi_0 p, \langle A_1' \nwarrow A_1 \rangle \pi_0 p) \sqsubseteq (\pi_0 \langle A_0' \times A_1' \nwarrow A_0 \times A_1 \rangle p, \pi_1 \langle A_0' \times A_1' \nwarrow A_0 \times A_1 \rangle p)}\ \mathcal{E}$$

$$\dfrac{\mathcal{E} \qquad (\pi_0 t, \pi_1 t) \sqsubseteq t \text{ with } t = \langle A_0' \times A_1' \nwarrow A_0 \times A_1 \rangle p}{p : A_0 \times A_1 \vdash (\langle A_0' \nwarrow A_0 \rangle \pi_0 p, \langle A_1' \nwarrow A_1 \rangle \pi_0 p) \sqsubseteq \langle A_0' \times A_1' \nwarrow A_0 \times A_1 \rangle p : A_0' \times A_1'}$$

Figure 7: Function, Product Upcast Reduction Derivations

Proof. We show that the second upcast is equivalent to the original in analogy with the way we show function/product types are unique: use the "elimination" rule of one and then the "introduction" rule of the other.

$$\dfrac{\dfrac{x : A \vdash x \sqsubseteq x : A}{x : A \vdash x \sqsubseteq \langle A' \nwarrow A \rangle x : A \sqsubseteq A'}\ \text{UR(S)'}}{x : A \vdash \langle A' \nwarrow A \rangle x \sqsubseteq \langle A' \nwarrow A \rangle x : A'}\ \text{UR(S)}$$

$$\dfrac{\dfrac{x : A \vdash x \sqsubseteq x : A}{x : A \vdash x \sqsubseteq \langle A' \nwarrow A \rangle x : A \sqsubseteq A'}\ \text{UR(S)}}{x : A \vdash \langle A' \nwarrow A \rangle x \sqsubseteq \langle A' \nwarrow A \rangle x : A'}\ \text{UR(S)'}$$

□

A dual proof gives uniqueness for downcasts as well.

Next, we show that the upcast and downcast for a given $A \sqsubseteq A'$ always form a *Galois Connection*. This fact forms the basis for our models considered in sections 5 and 6.

Theorem 3.6 (Casts are Galois Connections)**.** *If $A \sqsubseteq A'$, then $t \sqsubseteq \langle A \swarrow A' \rangle \langle A' \nwarrow A \rangle t$ and $\langle A' \nwarrow A \rangle \langle A \swarrow A' \rangle t \sqsubseteq t$*

Proof. The derivations are as follows:

$$\dfrac{\dfrac{x : A \vdash x \sqsubseteq x : A \sqsubseteq A}{x : A \vdash x \sqsubseteq \langle A' \nwarrow A \rangle x : A \sqsubseteq A'}\ \text{DR(S)}}{x : A \vdash x \sqsubseteq \langle A \swarrow A' \rangle \langle A' \nwarrow A \rangle x : A}\ \text{UL(S)}$$

$$\dfrac{\dfrac{x' : A' \vdash x' \sqsubseteq x' : A' \sqsubseteq A'}{x' : A' \vdash \langle A \swarrow A' \rangle x' \sqsubseteq x' : A \sqsubseteq A'}\ \text{UL(S)}}{x' : A' \vdash \langle A' \nwarrow A \rangle \langle A \swarrow A' \rangle x' \sqsubseteq x' : A'}\ \text{DL(S)}$$

□

In programming practice, we expect the stronger property that round trip from A to A' and back to be in fact an identity and this is implied by the addition of the *retract axiom*.

Recall that the gradual guarantee [33] says that making casts less dynamic results in semantically less dynamic terms, but does not otherwise change the behavior of programs. To see that a model of gradual type theory satisfies the gradual guarantee, the key syntactic fact is that making *casts* less dynamic results in a term dynamism relationship. We state this as the following theorem in GTT:

Theorem 3.7 (Cast Congruence)**.** *When $A \sqsubseteq A', B \sqsubseteq B', A \sqsubseteq B, A' \sqsubseteq B'$, we can derive*

$$x \sqsubseteq y : A \sqsubseteq B \vdash \langle A' \nwarrow A \rangle x \sqsubseteq \langle B' \nwarrow B \rangle y : A' \sqsubseteq B'$$

and

$$x' \sqsubseteq y' : A' \sqsubseteq B' \vdash \langle A \swarrow A' \rangle x' \sqsubseteq \langle B \swarrow B' \rangle y' : A \sqsubseteq B.$$

Proof. The proof of the first is

$$\dfrac{\dfrac{x \sqsubseteq y : A \sqsubseteq B \vdash x \sqsubseteq y : A \sqsubseteq B}{x \sqsubseteq y : A \sqsubseteq B \vdash x \sqsubseteq \langle B' \nwarrow B \rangle y : A \sqsubseteq B'}\ \text{UR(S)}}{x \sqsubseteq y : A \sqsubseteq B \vdash \langle A' \nwarrow A \rangle x \sqsubseteq \langle B' \nwarrow B \rangle y : A' \sqsubseteq B'}\ \text{UL(S)}$$

and the second is dual. □

All other term constructors are congruences by primitive rules, so $\sqsubseteq$ is a congruence.

Finally, we discuss what it means for two types to be equivalent in gradual type theory. Because types A and B in gradual type theory can be related both by type dynamism $A \sqsubseteq B$ and by functions $A \to B$, there are two reasonable notions of equivalence of types. First, if they are order-equivalent:

Definition 3.8 (Equi-dynamic)**.** We say types A and B are *equi-dynamic* if $A \sqsubseteq B$ and $B \sqsubseteq A$

And second, if they are isomorphic:

Definition 3.9 (Isomorphism)**.** We say types A and B are *isomorphic* if there exist terms $x : A \vdash t : B$ and $y : B \vdash u : A$ such that

$$x : A \vdash u[t/y] \sqsupseteq\sqsubseteq x : A$$

and

$$y : B \vdash t[u/x] \sqsupseteq\sqsubseteq y : B$$

Equi-dynamism of types turns out to be strictly stronger. First, equi-dynamism of types implies isomorphism, with the casts between them forming the isomorphism:

Theorem 3.10 (Equi-dynamic Types are Isomorphic)**.** *If $A \sqsupseteq\sqsubseteq B$, then*
(1) *$x.\langle B \nwarrow A\rangle x$ and $y.\langle A \nwarrow B\rangle y$ form an isomorphism of types.*
(2) *$y.\langle A \swarrow B\rangle x$ and $x.\langle B \swarrow A\rangle y$ form an isomorphism of types.*
(3) *$x.\langle B \nwarrow A\rangle x$ and $y.\langle A \swarrow B\rangle y$ form an isomorphism of types.*
(4) *$y : B \vdash \langle A \nwarrow B\rangle x \sqsupseteq\sqsubseteq \langle A \swarrow B\rangle x : A$.*

Proof. (1) By the following derivations:

$$\dfrac{\dfrac{x : A \vdash x \sqsubseteq x : A}{x : A \vdash \langle B \nwarrow A\rangle x \sqsubseteq x : B \sqsubseteq A}\ \mathrm{UL(S)}}{x : A \vdash \langle A \nwarrow B\rangle\langle B \nwarrow A\rangle x \sqsubseteq x : A}\ \mathrm{UL(S)}$$

$$\dfrac{\dfrac{x : A \vdash x \sqsubseteq x : A}{x : A \vdash x \sqsubseteq \langle B \nwarrow A\rangle x : A \sqsubseteq B}\ \mathrm{UR(S)}}{x : A \vdash x \sqsubseteq \langle A \nwarrow B\rangle\langle B \nwarrow A\rangle x : A}\ \mathrm{UR(S)}$$

(2) This is dual to the previous case.
(3) By the following derivation:

$$\dfrac{\dfrac{x : A \vdash x \sqsubseteq x : A}{x : A \vdash \langle B \nwarrow A\rangle x \sqsubseteq x : B \sqsubseteq A}\ \mathrm{UL(S)}}{x : A \vdash \langle A \swarrow B\rangle\langle B \nwarrow A\rangle x \sqsubseteq x : A}\ \mathrm{DL(S)}$$

$$\dfrac{\dfrac{x : A \vdash x \sqsubseteq x : A}{x : A \vdash x \sqsubseteq \langle B \nwarrow A\rangle x : A \sqsubseteq B}\ \mathrm{UR(S)}}{x : A \vdash x \sqsubseteq \langle A \swarrow B\rangle\langle B \nwarrow A\rangle x : A}\ \mathrm{DR(S)}$$

(4) This follows from uniqueness of inverses (which is true by the usual argument) and the previous two. □

The converse, that isomorphic types are equi-dynamic, does not hold by design, because it does not match gradual typing practice. Gradually typed languages typically have *disjointness* of connectives as operational reductions; for example, disjointness of products and functions can be expressed by an axiom $\langle (C \times D) \not\leftarrow ? \rangle \langle ? \not\leftarrow (A \to B) \rangle x \sqsubseteq \mathcal{U}$ which says that casting a function to a product errors. This axiom is incompatible with isomorphic types being equi-dynamic, because a function type *can* be isomorphic to a product type (e.g. $X \to Y \cong (X \to Y) \times 1$), and for equi-dynamic types A and B, a cast $\langle B \not\leftarrow ? \rangle \langle ? \not\leftarrow A \rangle x$ cannot fail, because if it fails, then every term of A, B equals $\mathcal{U}$: Assume $A \sqsupseteq\sqsubseteq B$ and $\langle B \not\leftarrow ? \rangle \langle ? \not\leftarrow A \rangle x \sqsubseteq \mathcal{U}$. By composition and the adjunction property

$$\langle B \not\leftarrow A \rangle x \sqsubseteq \langle B \not\leftarrow ? \rangle \langle ? \not\leftarrow B \rangle \langle B \not\leftarrow A \rangle x \sqsupseteq\sqsubseteq \langle B \not\leftarrow ? \rangle \langle ? \not\leftarrow A \rangle x \sqsubseteq \mathcal{U}$$

But by above, $\langle A \not\leftarrow B \rangle$ is an isomorphism, so

$$x : A \sqsupseteq\sqsubseteq \langle A \not\leftarrow B \rangle \langle B \not\leftarrow A \rangle x \sqsubseteq \langle B \not\leftarrow A \rangle \mathcal{U} \sqsupseteq\sqsubseteq \mathcal{U}$$

where the last step is by strictness, so every element of A (and B, by congruence of casts) is equal to a type error. That is, disjointness axioms make equi-dynamism an intensional property of the representation of a type, and therefore stronger than isomorphism. Nonetheless, the basic rules of gradual type theory do not imply disjointness; in Section 6, we discuss a countermodel.

4. Categorical Semantics

Next, we define what a category-theoretic model of preorder and gradual type theory is, and prove that PTT/GTT are *internal languages* of these classes of models by proving that they are left adjoint functors from categories of signatures to categories of models. This alternative axiomatic description of PTT/GTT is a useful bridge between the syntax and the concrete models presented in Section 6.

The models are based on a variant of *preorder categories*, which are categories internal to the category of preorders.[3] A preorder category is a category where the set of all objects and set of all arrows are each equipped with a preorder (a reflexive, transitive, but not necessarily anti-symmetric, relation). That is, rather than having merely a *set* of objects and *set* of arrows, preorder categories have a *preordered set* of objects and *preordered set* of arrows and the relevant functions are all monotone with respect to these orderings. A preorder category is equivalently a double category where one direction of morphism is thin. Intuitively, the preorder of objects represents types and type dynamism, while the preorder of morphisms represents terms and term dynamism, and we reuse the notation $\sqsubseteq$ for the orderings on objects and morphisms.

Definition 4.1 (Preorder Category)**.** A preorder category $\mathbb{C}$ consists of

(1) A preorder of "objects" $\mathbb{C}_0$
(2) A preorder of "arrows" $\mathbb{C}_1$
(3) Monotone functions of "source" and "target" $s, t : \mathbb{C}_1 \to \mathbb{C}_0$ and "identity" $i : \mathbb{C}_0 \to \mathbb{C}_1$

[3]To avoid confusion, these are not categories that happen to be preorders (thin categories) and these are not categories *enriched* in the category of preorders, where the hom-sets between two objects are preordered, but the set of objects is not.

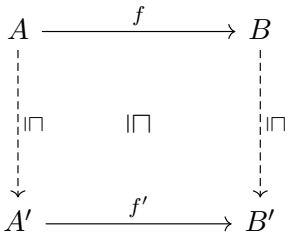
(4) A monotone composition function $\circ : \mathbb{C}_1 \times_{\mathbb{C}_0} \mathbb{C}_1 \to \mathbb{C}_1$ where $\mathbb{C}_1 \times_{\mathbb{C}_0} \mathbb{C}_1$ is the pullback of the source and target maps, which is explicitly given by the set

$$\mathbb{C}_1 \times_{\mathbb{C}_0} \mathbb{C}_1 = \{(f, g) \in \mathbb{C}_1^2 \mid sf = tg\}.$$

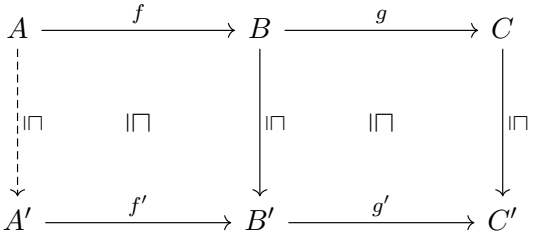
This composition must satisfy that for any $(f, g) \in \mathbb{C}_1 \times_{\mathbb{C}_0} \mathbb{C}_1$, we have $s(f \circ g) = sg$ and $t(f \circ g) = tf$.

(5) Unitality and associativity laws for composition: $f \circ i(A) = f$, $i(B) \circ f = f$ and $(f \circ g) \circ h = f \circ (g \circ h)$ whenever these are well-defined.

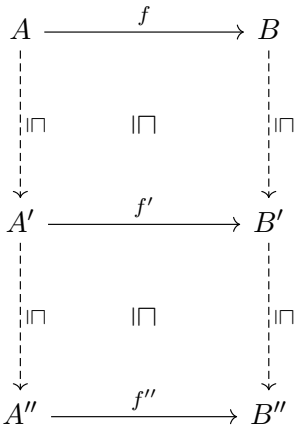
The algebra of composition in a preorder category can be viewed using a simple form of 2-dimensional string diagrams, which we adapt from [32]. We will not use this visualization for proofs, but we present it because they help to understand the somewhat complex presuppositions of the term dynamism judgment in gradual type theory. We draw the objects as points, and draw arrows horizontally, while drawing ordering relationships between objects vertically. Then the ordering relationship between arrows forms a kind of square, where the necessary ordering relationships are expressed geometrically. For instance if $f : A \to B$ is less than $f' : A' \to B'$, then it must be the case that $A \sqsubseteq A'$ and $B \sqsubseteq B'$ because source and target are monotone functions. In gradual type theory, this is part of the presuppositions of the term dynamism judgment. This somewhat complex relationship between domains and codomains is succinctly expressed as the following square which says that $f \sqsubseteq f'$:



Then, the substitution rule of gradual type theory can be visualized as horizontal concatenation of squares. If we have $f \sqsubseteq f'$ and $g \sqsubseteq g'$ then $g \circ f \sqsubseteq g' \circ f'$, which is visualized as



And the transitivity rule similarly corresponds to vertical concatenation of squares. So we visualize $f \sqsubseteq f' \sqsubseteq f''$ as



While the axioms of a preorder category are *similar to* the judgmental structure of preorder type theory, in a preorder category, morphisms have *one* source object and one target object, whereas in preorder type theory, terms have an entire *context* of inputs and one output. This is a standard mismatch between categories and type theories, and is often resolved by assuming that models have product types and using categorical products to interpret the context [17]. However, we will take a *multicategorical* view, in which our notion of model will axiomatize algebraically a notion of morphism with many inputs. Though for ordinary simple type theory the difference between the two is a matter of taste, for preorder type theory the difference is important when modeling term and specifically context dynamism. If a context $\Gamma = x_0 : A_0, \ldots$ is modeled as a product of its objects $A_0 \times \cdots$, then Γ should be less dynamic than another $\Gamma' = x_0' : A_0', \ldots$ just when their product is $A_0 \times \cdots \sqsubseteq A_0' \times \cdots$. However, in the syntax of our type theory, $\Gamma \sqsubseteq \Gamma'$ holds only when they are pointwise $\sqsubseteq$, i.e., $A_0 \sqsubseteq A_0', \ldots$ all hold. Since we allow for type dynamism axioms, these two notions are *not* equivalent, for instance there might be an axiom $1 \times 1 \sqsubseteq 1$, which would mean that $x : 1, y : 1 \sqsubseteq z : 1$ would hold if we interpreted contexts simply as their product. Therefore the syntax of context dynamism would be *incomplete* if it were interpreted as ordering of the products of the objects. Instead, we give a multicategorical definition in which the notion of context dynamism in the model is also pointwise. Specifically, we define a model of preorder type theory to be a *cartesian preorder multicategory*, to be defined shortly, which is like a preorder category that does not necessarily have true product *objects*, but whose morphisms' source can be a "virtual" product of objects, i.e. a context.

We developed this notion of cartesian preorder multicategory using the theory of generalized multicategories [6]. Briefly, there is a monad on the (double) category of preorder categories whose algebras are preorder categories with cartesian products and cartesian preorder multicategories are the generalized multicategories with respect to this monad. While we will not make use of this abstract formulation, it was quite useful in ensuring we had a reasonable definition.

Definition 4.2 (CPM)**.** We mutually define what constitutes a cartesian preorder multicategory (CPM) $\mathbb{C}$ with an associated preorder category $\mathrm{Ctx}(\mathbb{C})$ of "contexts" and "substitutions".

- A cartesian preorder multicategory (CPM) $\mathbb{C}$ consists of
  (1) a preordered set of "objects" $\mathbb{C}_0$

(2) a preordered set of "multiarrows" $\mathbb{C}_1$
(3) Monotone functions "source" $s : \mathbb{C}_1 \to \mathrm{Ctx}(\mathbb{C})_0$ and "target" $t : \mathbb{C}_1 \to \mathbb{C}_0$. We define $\mathbb{C}_1(\Gamma; A) = \{f \in \mathbb{C}_1 \mid s(f) = \Gamma \wedge t(f) = A\}$
(4) Monotone "projection" functions $x : \mathrm{Ctx}(\mathbb{C})_0 \times \mathbb{C}_0 \times \mathrm{Ctx}(\mathbb{C})_0 \to \mathbb{C}_1$ satisfying $s(x(\Gamma; A; \Delta)) = \Gamma, A, \Delta$ and $t(x(\Gamma; A; \Delta)) = A$. We will sometimes refer to $(x(\cdot; A; \cdot)) \in \mathbb{C}_1(A; A)$ as id_A, the identity multiarrow.
(5) A monotone "composition" function $\circ : \mathbb{C}_1 \times_{\mathrm{Ctx}(\mathbb{C})_0} \mathrm{Ctx}(\mathbb{C})_1 \to \mathbb{C}_1$ satisfying $s(f \circ \gamma) = s(\gamma)$ and $t(f \circ \gamma) = t(f)$ where $\mathbb{C}_1 \times_{\mathrm{Ctx}(\mathbb{C})_0} \mathrm{Ctx}(\mathbb{C})_1$ denotes a pullback, which is explicitly given by
$$\mathbb{C}_1 \times_{\mathrm{Ctx}(\mathbb{C})_0} \mathrm{Ctx}(\mathbb{C})_1 = \{(f, \gamma) \mid t(\gamma) = s(f)\}$$
equipped with the pointwise ordering.
(6) Satisfying the "Identity" law: for any $f \in \mathbb{C}_1(\Gamma; A)$,
$$f \circ \mathrm{id}_\Gamma = f$$
(7) Satisfying the "Projection" law, that for every $\gamma \in \mathrm{Ctx}(\mathbb{C})_1(\Theta; \Gamma, A, \Delta)$,
$$x(\Gamma; A; \Delta) \circ \gamma = \gamma(|\Gamma|)$$
(8) Satisfying the "Associativity" law: for any $f \in \mathbb{C}_1(\Gamma; A)$, $\gamma \in \mathrm{Ctx}(\mathbb{C})_1(\Delta; \Gamma)$, $\delta \in \mathrm{Ctx}(\mathbb{C})_1(\Theta; \Delta)$
$$(f \circ \gamma) \circ \delta = f \circ (\gamma \circ \delta)$$

- $\mathrm{Ctx}(\mathbb{C})$ is a preorder category where
(1) Objects $\Gamma \in \mathrm{Ctx}(\mathbb{C})_0$ are lists of elements of $\mathbb{C}_0$ (called "contexts") with the pointwise ordering.
(2) We define $\mathrm{Ctx}(\mathbb{C})_1(\Delta; \Gamma)$, the set of morphisms with source Δ and target $\Gamma = A_0, \ldots$ to consist of functions γ that assign for every $i \in \{0, \ldots, |\Gamma| - 1\}$ a multiarrow $\gamma(i) \in \mathbb{C}_1(\Delta; A_i)$. Then $\mathrm{Ctx}(\mathbb{C})_1$ is the set of triples $(\gamma; \Delta; \Gamma)$ such that $\gamma \in \mathrm{Ctx}(\mathbb{C})_1(\Delta; \Gamma)$, equipped with the pointwise ordering.
(3) Given $\gamma \in \mathrm{Ctx}(\mathbb{C})_1(\Delta; \Gamma)$ and $\gamma' \in \mathrm{Ctx}(\mathbb{C})_1(\Gamma; A_1, \ldots, A_n)$, we define the composition
$$(\gamma' \circ \gamma)(i) = \gamma'(i) \circ \gamma$$
where the latter $\circ$ is composition of a substitution with a multiarrow.
(4) For an object $\Gamma = A_0, \ldots, A_n$, the identity substitution is given by the pointwise projections morphism.
$$\mathrm{id}_\Gamma = (x(\cdot; A_0; A_1, \ldots, A_n), \ldots, x(A_0, \ldots, A_{n-1}; A_n; \cdot))$$

An intuition for the axioms of multiarrow composition with substitutions are that they are precisely what is needed to make the definition of identity and composition for $\mathrm{Ctx}(\mathbb{C})$ into a cartesian preorder category where the cartesian product is given by context concatenation.

4.1. **Soundness, Completeness and Initiality for PTT.** Next, we present the soundness and completeness theorems of the interpretation of preorder type theory in a cartesian preorder multicategory. Soundness informally means that any interpretation of the base types and function symbols in a CPM that satisfy the axioms in a signature can be extended to a compositional semantics in which all derivable type and term dynamism theorems are true in the model. To make this precise, we first need to define precisely what a valid

interpretation of a signature in a CPM is. This proceeds in stages. First we define an interpretation of base types.

Definition 4.3 (Interpretation of Base Types)**.** An interpretation of a 0-PTT signature Σ_0 in a CPM $\mathbb{C}$ is a function $(\!|\cdot|\!)_0 : \Sigma_0 \to \mathbb{C}_0$ assigning an object to each base type.

Then we define when an interpretation of base types validates all type dynamism axioms.

Definition 4.4 (Interpretation of Type Dynamism Axioms)**.** An interpretation $(\!|\cdot|\!)_0$ of Σ_0 in $\mathbb{C}$ extends to an interpretation of a 1-signature Σ_1, if for every type dynamism axiom $(A, B) \in \Sigma_1$, $(\!|A|\!)_0 \sqsubseteq (\!|B|\!)_0$

And we can easily prove that any interpretation of type dynamism axioms extends to a *sound* interpretation of type dynamism proofs.

Theorem 4.5 (Soundness of PTT Type Dynamism)**.** *If $(\!|\cdot|\!)_0$ is an interpretation of Σ_0, Σ_1 in $\mathbb{C}$, then if $A \sqsubseteq A'$ is derivable then $(\!|A|\!)_0 \sqsubseteq (\!|A'|\!)_0$ in $\mathbb{C}$.*

Proof. By induction on proofs of $A \sqsubseteq A'$. Reflexivity and transitivity follows by the fact that $\sqsubseteq$ is a preorder and axioms follow by assumption that $(\!|\cdot|\!)_0$ is an interpretation of Σ_1. □

Then we define an interpretation of function symbols that extends an interpretation of base types and type dynamism axioms.

Definition 4.6 (Interpretation of Function Symbols)**.** If $(\!|\cdot|\!)_0$ is an interpretation of Σ_0, Σ_1 in a CPM $\mathbb{C}$, and Σ_2 is 2-PTT signature extending Σ_0, Σ_1 then an extension of $(\!|\cdot|\!)_0$ to interpret function symbols is a function $(\!|\cdot|\!)_2 : \Sigma_2 \to \mathbb{C}_1$ that respects typing in that $t((\!|f|\!)_2) = (\!|t(f)|\!)_0$ and $s((\!|f|\!)_2) = (\!|s(f)|\!)_0$, where the last formula means the pointwise application of $(\!|\cdot|\!)_0$ to each element of the list of input types.

Next, we define how to extend such an interpretation to an interpretation of all PTT terms.

Definition 4.7 (Interpretation of PTT Terms)**.** If $(\!|\cdot|\!)_0, (\!|\cdot|\!)_1$ are interpretations of $\Sigma_0, \Sigma_1, \Sigma_2$ in a CPM $\mathbb{C}$, we extend this to an interpretation $[\![\cdot]\!]_2$ of all PTT terms in $\mathbb{C}$ as follows.

$$[\![\Gamma \vdash f(t_0, \ldots)]\!]_2 = (\!|f|\!) \circ ([\![t_0]\!]_2, \ldots)$$
$$[\![\Gamma, x : A, \Delta \vdash x : A]\!]_2 = x((\!|\Gamma|\!)_0; (\!|A|\!)_0; (\!|\Delta|\!)_0)$$

where we define $(\!|x_0 : A_0, \ldots|\!)_0 = (\!|A_0|\!)_0, \ldots$. Note that this respects typing in that if $\Gamma \vdash t : A$, then $s([\![t]\!]_2) = (\!|\Gamma|\!)_0$ and $t([\![t]\!]_2) = (\!|A|\!)_0$.

Finally, we define when an interpretation satisfies all *term* dynamism axioms.

Definition 4.8 (Interpretation of GTT Term Dynamism Axioms)**.** If $(\!|\cdot|\!)_0, (\!|\cdot|\!)_2$ form an interpretation of $\Sigma_0, \Sigma_1, \Sigma_2$ in $\mathbb{C}$ and Σ_3 is a 3-PTT signature relative to $\Sigma_0, \Sigma_1, \Sigma_2$, then we say $(\!|\cdot|\!)_0, (\!|\cdot|\!)_1$ interpret Σ_3 if for every term dynamism axiom $(t, u) \in \Sigma_3$, $[\![t]\!]_2 \sqsubseteq [\![u]\!]_2$ holds in $\mathbb{C}$.

Then we can show that this interpretation of term dynamism is also *sound* in that if all term dynamism axioms are valid, then all derivable term dynamism statements are true in the model.

Theorem 4.9 (Soundness of Term Dynamism)**.** *If $(\!|\cdot|\!)_0, (\!|\cdot|\!)_2$ form an interpretation of $\Sigma = (\Sigma_0, \Sigma_1, \Sigma_2, \Sigma_3)$ in a CPM $\mathbb{C}$, then for every $\Phi \vdash t \sqsubseteq t' : A \sqsubseteq A'$ provable in PTT from Σ, $[\![t]\!]_2 \sqsubseteq [\![u]\!]_2$ holds in $\mathbb{C}$.*

Proof. By induction on term dynamism derivations.

(1) Variable rule follows by monotonicity of projections
(2) Composition rule follows by monotonicity of $\circ$.
(3) Reflexivity and transitivity follow because $\sqsubseteq$ is a preorder in $\mathbb{C}$.
(4) Axioms follow by assumption. □

We summarize these in the following soundness theorem.

Theorem 4.10 (Soundness of Preorder Type Theory)**.** *For any PTT signature Σ and CPM $\mathbb{C}$ and interpretation $(\!|\cdot|\!)_0, (\!|\cdot|\!)_1$ of Σ in $\mathbb{C}$,*

(1) *For every PTT type A, $[\![A]\!]_0$ is an object of $\mathbb{C}$.*
(2) *If $A \sqsubseteq A'$ is provable in PTT, then $[\![A]\!]_0 \sqsubseteq [\![A']\!]_0$ in $\mathbb{C}$.*
(3) *For every $\Gamma \vdash t : A$ in PTT, $[\![t]\!]_2$ is a multi-arrow in $\mathbb{C}$ with $s([\![t]\!]_2) = (\!|\Gamma|\!)_0$ and $t([\![t]\!]_2) = (\!|A|\!)_0$.*
(4) *If $\Phi \vdash t \sqsubseteq t' : A \sqsubseteq A'$ is provable in GTT, then $[\![t]\!]_2 \sqsubseteq [\![t']\!]_2$ in $\mathbb{C}$.*

Next, we show the *completeness* of PTT with respect to this semantics, which informally means that if a type or term dynamism ordering is satisfied in every CPM that interprets a signature, then the ordering is syntactically derivable. We prove it in the standard method for categorical models, which is to show that the *syntax itself* presents a CPM where term and type dynamism are given by derivability of syntactic term and type dynamism proofs.

Theorem 4.11 (Completeness of CPM Semantics)**.** *Let Σ be a PTT signature and let all syntax be relative to Σ.*

(1) *For any two types A, A', if $(\!|A|\!)_0 \sqsubseteq (\!|A'|\!)_0$ for every interpretation $(\!|\cdot|\!)_0, (\!|\cdot|\!)_1$ of Σ, then $A \sqsubseteq A'$ is derivable in PTT.*
(2) *For any two terms t, t', if $[\![t]\!] \sqsubseteq [\![t']\!]$ for every interpretation $(\!|\cdot|\!)_0, (\!|\cdot|\!)_2$, then $t \sqsubseteq t'$ is derivable in PTT.*

Proof. We construct the CPM PTT(Σ) as follows:

(1) The objects are the types generated by Σ.
(2) $A \sqsubseteq A'$ holds when $A \sqsubseteq A'$ is derivable.
(3) A term $\mathrm{PTT}(\Sigma)(A_0, \ldots; B)$ is a term $x_0 : A_0, \ldots \vdash t : B$ for some variables $x_0, \ldots,$, quotiented by α-renaming (but not reordering). Composition is given by substitution and identity/projection by variable usage.
(4) $t \sqsubseteq t'$ holds when $\Phi \vdash t \sqsubseteq t' : A \sqsubseteq A'$ for the unique Φ, A, A' making that well-formed.

Proving this is a CPM involves the standard proofs of the associativity and unitality of substitution and an easy proof that substitution is monotone with respect to term dynamism.

There is an obvious interpretation of Σ in PTT(Σ) that interprets every base type and function symbol as itself. Then if a type/term dynamism theorem is true in every model, it is in particular true for PTT(Σ), which means exactly that it is derivable. □

Together these theorems imply *initiality* of Preorder Type Theory in the category of cartesian preorder multicategories, analogous to the classical theorem for cartesian closed categories and typed λ-calculus [17]. Essentially the initiality theorem packages up soundness and completeness into a compact abstract statement, and formalizes the way in which preorder type theory is a canonical syntax for CPMs.

First, we define a category of signatures and signature translations.

Definition 4.12 (Category of PTT Signatures)**.** We define a translation of PTT signatures $i : \Sigma \to \Theta$ consists of

(1) a function translating base types $i_0 : \Sigma_0 \to \Theta_0$.
(2) and a function translating function symbols $i_2 : \Sigma_2 \to \Theta_2$ that respects typing in that $t(i_2(f)) = i_0(t(f))$ and $s(i_2(f)) = i_0(s(f))$ where $i_0(s(f))$ is the pointwise application of i_0.

Such that axioms are translated to axioms in that

(1) if $(X, Y) \in \Sigma_1$, then $(i_0(X), i_0(Y)) \in \Theta_1$
(2) if $(f, g) \in \Sigma_3$, then $(i_2(f), i_2(g)) \in \Theta_3$.

Translations compose by composing the components and this makes a category PttS whose objects are PTT signatures and morphisms are translations.

Then we define a category of CPMs and functors.

Definition 4.13 (Category of CPMs)**.** A functor of CPMs $F : \mathbb{C} \to \mathbb{D}$ consists of a monotone function on objects $F_0 : \mathbb{C}_0 \to \mathbb{D}_0$ and a monotone function on multiarrows $F_1 : \mathbb{C}_1 \to \mathbb{D}_1$ that preserves source and target in that $t(F_1(f)) = F_0(t(f)), s(F_1(f)) = F_0(s(f))$, preserves composition in that $F_1(f \circ \gamma) = F_1(f) \circ F_1(\gamma)$ and preserves projection in that $F_1(x(\Gamma; A; \Delta)) = x(F_1(\Gamma); F_1(A); F_1(\Delta))$. Note that we implicitly have lifted F_0, F_1 pointwise to apply to contexts and substitutions

This makes a category CPM whose objects are CPMs and morphisms are functors with obvious identity and composition.

Next, we define a functor that takes a CPM and produces the "complete signature": one that includes all types as base types, all multi-arrows as function symbols, and all true dynamism facts as axioms.

Definition 4.14 (Complete Signature)**.** We define the complete signature functor $U : CPM \to$ PttS on objects as follows.

(1) $(U(\mathbb{C}))_0 = C_0$
(2) $(U(\mathbb{C}))_1 = \{(A, A') \in \mathbb{C}_1^2 \mid A \sqsubseteq A'\}$
(3) $(U(\mathbb{C}))_2 = \mathbb{C}_1$
(4) $(U(\mathbb{C}))_3 = \{(f(x_0, \ldots), g(x_0, \ldots)) \mid f, g \in \mathbb{C}_1 \wedge f \sqsubseteq g \wedge |s(f)| = n\}$

$U(\mathbb{C})$ has as base types the objects of $\mathbb{C}$, type dynamism axioms all true orderings between objects in $\mathbb{C}$, function symbols all morphisms of $U(\mathbb{C})$ and term dynamism axioms all true orderings of morphisms in $\mathbb{C}$. This clearly extends to a functor.

Then we can prove that model construction is a left adjoint functor to this forgetful functor, establishing that the construction in the proof of the completeness theorem produces the "minimal" way to make a CPM from a signature. The connection to the soundness and completeness theorems can be seen by the fact that an interpretation $(\!|\cdot|\!)_0, (\!|\cdot|\!)_1$ of a signature Σ in a CPM $\mathbb{C}$ is equivalent to a translation of signatures $\Sigma \to U(\mathbb{C})$. The initiality theorem then states that any such interpretation uniquely extends to a morphism of CPMs $PTT(\Sigma) \to \mathbb{C}$.

Theorem 4.15 (PTT is the Initial CPM generated by a Signagture)**.** *The syntax of preorder type theory $PTT : PttS \to CPM$ is a left adjoint functor to U.*

Proof. We use the formulation of a left adjoint in terms of universal morphisms [19]. Note that this definition does not presuppose that PTT is a functor, but rather proves that it is a functor.

(1) As shown in the completeness theorem 4.11, $PTT(\Sigma)$ is a CPM.
(2) For any signature Σ, there is a "universal" translation of signatures $\eta_\Sigma : \Sigma \to U(PTT(\Sigma))$ that interprets every generator as "itself", i.e.

$$\eta(X \in \Sigma_0) = X$$
$$\eta(f \in \Sigma_2) = f(x_0, \ldots, x_{n-1})$$

where n is the length of $s(f)$. Every ordering axiom is satisfied in $U(PTT(\Sigma))$ by definition of U.
(3) For every translation $(\!|\cdot|\!) : \Sigma \to U\mathbb{C}$, there is a morphism of CPMs $[\![\cdot]\!] : PTT(\Sigma) \to \mathbb{C}$ given by setting $[\![A]\!]_0 = (\!|A|\!)_0$ and defining $[\![t]\!]_2$ as in the proof of the soundness Theorem 4.7. The interpretation is functorial because it satisfies

$$[\![t[\gamma]]\!]_2 = [\![t]\!]_2 \circ [\![\gamma]\!]_2$$

which follows by induction.
(4) Every translation $(\!|\cdot|\!) : \Sigma \to U\mathbb{C}$ factors through the universal translation in that $(\!|\cdot|\!) = U([\![\cdot]\!]) \circ \eta_\Sigma$. First, for any base type $X \in \Sigma_0$, we have

$$[\![\eta_\Sigma(X)]\!]_0 = [\![X]\!]_0 = (\!|X|\!)_0$$

Next, for any function symbol $f \in \Sigma_1$ with input types $A_0, \ldots$, we have

$$\begin{aligned}
[\![\eta_\Sigma(f)]\!]_2 &= [\![f(x_0, \ldots)]\!]_2 \\
&= (\!|f|\!)_2 \circ ([\![x_0]\!]_2, \ldots) \\
&= (\!|f|\!)_2 \circ \mathrm{id}_{([\![A_0]\!]_0, \ldots)} \\
&= (\!|f|\!)_2
\end{aligned}$$

(5) $[\![\cdot]\!]$ is the *unique* morphism satisfying this factorization. To show this, let $F : PTT(\Sigma) \to \mathbb{C}$ be a CPM morphism satisfying $UF \circ \eta_\Sigma = i$. We need to show that $F = [\![\cdot]\!]$. First, on objects, because of the factorization

$$F_0(X) = (\!|X|\!)_0 = [\![X]\!]_0$$

Next, for morphisms we proceed by induction on PTT terms.
(a) For a variable $\Gamma, x : A, \Delta \vdash x : A$,

$$\begin{aligned}
F_1(x) &= x(F_0(\Gamma); F_0(A); F_0(\Delta)) && \text{(functoriality)} \\
&= x([\![\Gamma]\!]_0; [\![A]\!]_0; [\![\Delta]\!]_0) && \text{(factorization on objects)} \\
&= [\![x]\!]_2 && \text{(definition)}
\end{aligned}$$

(b) For function applications

$$\begin{aligned}
F_1(f(t_0, \ldots)) &= F_1(f) \circ (F_1(t_0), \ldots) && \text{(functoriality)} \\
&= (\!|f|\!)_2 \circ (F_1(t_0), \ldots) && \text{(factorization)} \\
&= (\!|f|\!)_2 \circ ([\![t_0]\!]_2, \ldots) && \text{(induction)} \\
&= [\![f(t_0, \ldots)]\!]_2 && \text{(definition)}
\end{aligned}$$

□

4.2. **Gradual Typing Structures.** Next, we describe the additional structure on a CPM to model full gradual type theory: casts are modeled by the structure of an *equipment* [32], a dynamic type by a greatest object, and the type error by a least element of every hom-set.

Definition 4.16 (Upcasts, Downcasts in a CPM)**.** In a CPM $\mathbb{C}$, if $A \sqsubseteq A'$, we define

(1) A morphism $u : A \to A'$ is an *upcast* for $A \sqsubseteq A'$ if $u \sqsubseteq \mathrm{id}_{A'}$ and $\mathrm{id}_A \sqsubseteq u$.
(2) A morphism $d : A' \to A$ is a *downcast* for $A \sqsubseteq A'$ if $d \sqsubseteq \mathrm{id}_{A'}$ and $\mathrm{id}_A \sqsubseteq d$.

It will be useful for the completeness theorem that upcasts and downcasts are unique up to order-equivalence, a semantic version of Theorem 3.5.

Lemma 4.17 (Upcasts, Downcasts are Unique)**.** *If u, u' are both upcasts for $A \sqsubseteq A'$, then $u \sqsupseteq\sqsubseteq u'$. Similarly if d, d' are both downcasts for $A \sqsubseteq A'$, then $d \sqsupseteq\sqsubseteq d'$.*

Proof. By duality it is sufficient to show the upcast case. By symmetry of the situation it is sufficient to show $u \sqsubseteq u'$. First, $u = u \circ \mathrm{id}_A$ and $u' = \mathrm{id}_{A'} \circ u'$. Since they are upcasts, we also have $u \sqsubseteq \mathrm{id}_{A'}$ and $\mathrm{id}_A \sqsubseteq u'$, so by monotonicity of composition we have $u = u \circ \mathrm{id}_A \sqsubseteq \mathrm{id}_{A'} \circ u' = u'$. □

Definition 4.18 (Equipment [32])**.** A CPM $\mathbb{C}$ is an *equipment* if for every $A \sqsubseteq A'$, there exist upcasts and downcasts for $A \sqsubseteq A'$. An equipment is *coreflective* if also $d_{A,A'} \circ u_{A,A'} \sqsubseteq \mathrm{id}_A$.

Definition 4.19 (Greatest Object)**.** A greatest object $\top$ in a CPM $\mathbb{C}$ is a greatest element of the preorder of objects $\mathbb{C}_0$.

Definition 4.20 (Local Bottoms)**.** A CPM $\mathbb{C}$ *has local bottoms* if every hom set $\mathbb{C}(\Gamma; A)$ has a least element $\bot_{\Gamma;A}$ and for every substitution $\gamma \in \mathrm{Ctx}(\mathbb{C})_1(\Delta; \Gamma)$ we have $\bot_{\Gamma;A} \circ \gamma \sqsupseteq\sqsubseteq \bot_{\Delta;A}$.

4.3. **Interpreting Negative Types.** Next, we define a *cartesian closed CPM*, which will model negative function and product types. While we use the adjectives "closed" and "cartesian", the structure exhibited here is unique up to canonical *isomorphism*, but *not* unique up to *order-equivalence* (equi-dynamism). Thus, there may be multiple order-inequivalent ways that a CPM can be closed or cartesian, so it is important that e.g. a closed CPM is a CPM *with* a *choice* of exponentials. Since all of the concrete models we consider are strict, we take a strict interpretation of naturality and $\beta\eta$, but this could likely be weakened.

Definition 4.21 (Closed CPM)**.** A Closed CPM is a CPM $\mathbb{C}$ with a monotone function on objects $\to\colon \mathbb{C}_0^2 \to \mathbb{C}_0$ making for every pair of objects $A, B \in \mathbb{C}$ an "exponential" object $A \to B$ with a monotone function

$$\lambda : \mathbb{C}(\Gamma, A; B) \to \mathbb{C}(\Gamma; A \to B)$$

that is *natural* in that for any appropriate Γ, γ, h

$$\lambda(h) \circ \gamma = \lambda(h \circ (\gamma, x(\Gamma; A; \cdot)))$$

with a morphism

$$\mathrm{app} \in \mathbb{C}(A \to B, A; B)$$

such that the function given by

$$f \mapsto \mathrm{app} \circ (f \circ \mathrm{wkn}, x(\Gamma; A; \cdot)) : \mathbb{C}(\Gamma; A \to B) \to \mathbb{C}(\Gamma, A; B)$$

is an inverse to λ (all up to equality), where here $\mathrm{wkn} \in \mathrm{Ctx}(\mathbb{C})_1(\Gamma, A; \Gamma)$ is the evident weakening substitution.

Definition 4.22 (Cartesian CPM)**.** A Cartesian CPM is a CPM $\mathbb{C}$ with a monotone function $\times : \mathbb{C}_0^2 \to \mathbb{C}_0$ and a chosen object $1 \in \mathbb{C}_0$ with functions

$$\text{pair} : \mathbb{C}(\Gamma; A) \times \mathbb{C}(\Gamma; B) \to \mathbb{C}(\Gamma; A \times B)$$
$$\text{unit} : 1 \to \mathbb{C}(\Gamma; 1)$$

that are natural in that for any f, g, γ

$$\text{pair}(f, g) \circ \gamma = \text{pair}(f \circ \gamma, g \circ \gamma)$$
$$\text{unit} \circ \gamma = \text{unit}$$

and morphisms

$$\pi_1 : \mathbb{C}(A \times B; A) \quad \pi_2 : \mathbb{C}(A \times B; B)$$

such that the function given by

$$f \mapsto (\pi_1 \circ f, \pi_2 \circ f) : \mathbb{C}(\Gamma; A \times B) \to \mathbb{C}(\Gamma; A) \times \mathbb{C}(\Gamma; B)$$

is an inverse to pair.

A *cartesian closed CPM* is a CPM with a choice of both cartesian and closed structure.

Definition 4.23. A *GTT category* is a CPM that is cartesian closed, a coreflective equipment and has a greatest object and local bottoms.

4.4. **Soundness, Completeness and Initiality of GTT.** Next, we extend the interpretation of preorder type theory to gradual type theory. Note that here the interpretation of casts depends on the soundness of type dynamism, since we need to know that dynamism has a sound interpretation in order for the semantic upcasts and downcasts to exist. So we proceed in stages: first an interpretation of base types, then type dynamism, then terms and finally term dynamism.

Definition 4.24 (Interpretation of Base Types)**.** An interpretation of a 0-GTT signature Σ_0 in a GTT category $\mathbb{C}$ is a function $(\!|\cdot|\!)_0 : \Sigma_0 \to \mathbb{C}_0$ assigning an object to each base type.

Definition 4.25 (Interpretation of GTT Types, Contexts)**.** Given an interpretation $(\!|\cdot|\!)_0$ of Σ_0 in a GTT category $\mathbb{C}$, we extend this to an interpretation $[\![\cdot]\!]_0$ of all types generated from Σ_0 as follows:

$$[\![X]\!]_0 = (\!|X|\!)_0$$
$$[\![?]\!]_0 = \top$$
$$[\![A \to B]\!]_0 = [\![A]\!]_0 \to [\![B]\!]_0$$
$$[\![A \times B]\!]_0 = [\![A]\!]_0 \times [\![B]\!]_0$$
$$[\![1]\!]_0 = 1$$

We extend this also to an interpretation of contexts by defining

$$[\![x_0 : A_0, \ldots]\!]_0 = [\![A_0]\!]_0, \ldots$$

Definition 4.26 (Interpretation of Type Dynamism Axioms)**.** An interpretation $(\!|\cdot|\!)_0$ of Σ_0 in $\mathbb{C}$ extends to an interpretation of a 1-GTT signature Σ_1 if for every type dynamism axiom $(A, B) \in \Sigma_1$, $[\![A]\!]_0 \sqsubseteq [\![B]\!]_0$.

Theorem 4.27 (Soundness of GTT Type Dynamism)**.** *If $(\!|\cdot|\!)_0$ is an interpretation of Σ_0, Σ_1 in $\mathbb{C}$, then if $A \sqsubseteq A'$ is provable in GTT from Σ_0, Σ_1, then $[\![A]\!]_0 \sqsubseteq [\![A']\!]_0$*

Proof. By induction on type dynamism derivations.

(1) The reflexivity and transitivity cases follow by the fact that $\sqsubseteq$ in $\mathbb{C}$ is a preorder.
(2) $[\![A]\!]_0 \sqsubseteq [\![?]\!]_0$ holds because $[\![?]\!]_0 = \top$, which is a greatest element.
(3) The function and product cases follow because exponentials and products in $\mathbb{C}$ are assumed monotone. □

Definition 4.28 (Interpretation of Function Symbols)**.** If $(\!|\cdot|\!)_0$ is an interpretation of Σ_0, Σ_1 in $\mathbb{C}$, and Σ_2 is 2-GTT signature extending Σ_0, Σ_1 then an extension of $(\!|\cdot|\!)_0$ to interpret function symbols is a function $(\!|\cdot|\!)_2 : \Sigma_2 \to \mathbb{C}_1$ that respects typing in that $t((\!|f|\!)_2) = [\![t(f)]\!]_0$ and $s((\!|f|\!)_2) = [\![s(f)]\!]_0$.

Definition 4.29 (Interpretation of GTT Terms, Substitutions)**.** If $(\!|\cdot|\!)_0, (\!|\cdot|\!)_2$ form an interpretation of $\Sigma_0, \Sigma_1, \Sigma_2$ in $\mathbb{C}$, then we extend this to an interpretation $[\![\cdot]\!]_2$ of all terms in GTT generated by $\Sigma_0, \Sigma_1, \Sigma_2$ as follows:

$$
\begin{aligned}
[\![f(t_0, \ldots)]\!]_2 &= (\!|f|\!)_2 \circ ([\![t_0]\!]_2, \ldots) \\
[\![\Gamma, x : A, \Delta \vdash x : A]\!]_2 &= x([\![\Gamma]\!]_0; [\![A]\!]_0; [\![\Delta]\!]_0) \\
[\![\langle A' \nwarrow A \rangle t]\!]_2 &= u_{[\![A]\!]_0, [\![A']\!]_0} \circ ([\![t]\!]_2) \\
[\![\langle A \swarrow A' \rangle t]\!]_2 &= d_{[\![A]\!]_0, [\![A']\!]_0} \circ ([\![t]\!]_2) \\
[\![\mho]\!]_2 &= \bot \\
[\![\lambda x : A.t]\!]_2 &= \lambda([\![t]\!]_2) \\
[\![tu]\!]_2 &= \mathrm{app} \circ ([\![t]\!]_2, [\![u]\!]_2) \\
[\![(t_1, t_2)]\!]_2 &= \mathrm{pair}([\![t_1]\!]_2, [\![t_2]\!]_2) \\
[\![\pi_i t]\!]_2 &= \pi_i \circ ([\![t]\!]_2) \\
[\![()]\!]_2 &= \mathrm{unit}
\end{aligned}
$$

And we extend it to substitutions by defining $[\![\gamma]\!]_2(i) = [\![\gamma(x_i)]\!]_2$ for $\gamma : \Delta \vdash x_0 : A_0, \ldots$.

Definition 4.30 (Interpretation of Term Dynamism Axioms)**.** If $(\!|\cdot|\!)_0, (\!|\cdot|\!)_2$ form an interpretation of $\Sigma_0, \Sigma_1, \Sigma_2$ in $\mathbb{C}$ and Σ_3 is a 3-GTT signature relative to $\Sigma_0, \Sigma_1, \Sigma_2$, then we say $(\!|\cdot|\!)_0, (\!|\cdot|\!)_1$ interpret Σ_3 if for every term dynamism axiom $(t, u) \in \Sigma_3$, $[\![t]\!]_2 \sqsubseteq [\![u]\!]_2$.

Theorem 4.31 (Soundness of GTT Term Dynamism)**.** *If $(\!|\cdot|\!)_0, (\!|\cdot|\!)_2$ form an interpretation of $\Sigma = (\Sigma_0, \Sigma_1, \Sigma_2, \Sigma_3)$, then for every $\Phi \vdash t \sqsubseteq t' : A \sqsubseteq A'$ provable in GTT from Σ, $[\![t]\!]_2 \sqsubseteq [\![t']\!]_2$.*

Proof. By induction on term dynamism derivations.

(1) The variable, composition, reflexivity, transitivity and axiom rules follow b the same argument as for PTT.
(2) The upcast and downcast rules follow by definition of an equipment.
(3) The type error rule $\mho \sqsubseteq t$ follows by definition of a local bottom.
(4) For the types $\to, \times, 1$, the congruence rules hold by monotonicity of the cartesian and closed structures, and the $\beta\eta$ by the equational laws for the closed structure. □

We summarize these results in the following Soundness theorem

Theorem 4.32 (Soundness of Gradual Type Theory)**.** *For any GTT signature Σ and GTT category $\mathbb{C}$ and interpretation $(\!|\cdot|\!)$ of Σ in $\mathbb{C}$,*

(1) *For every GTT type A, $[\![A]\!]_0$ is an object of $\mathbb{C}$.*

(2) *If $A \sqsubseteq A'$ is provable in GTT, then $[\![A]\!]_0 \sqsubseteq [\![A']\!]_0$.*
(3) *For every $\Gamma \vdash t : A$ in GTT, $[\![t]\!]_2$ is a multi-arrow in $\mathbb{C}$ with $s([\![t]\!]_2) = [\![\Gamma]\!]_0$ and $t([\![t]\!]_2) = [\![A]\!]_0$ where $[\![x_0 : A_0, \ldots]\!]_0 = [\![A_0]\!]_0 \ldots$.*
(4) *If $\Phi \vdash t \sqsubseteq t' : A \sqsubseteq A'$ is provable in GTT, then $[\![t]\!]_2 \sqsubseteq [\![t']\!]_2$.*

Next, we have the *completeness* theorem for GTT which says that if a dynamism is true in every interpretation of a signature, then it is provable in GTT.

Theorem 4.33 (Completeness of GTT Category Semantics)**.** *Let Σ be a GTT signature.*
(1) *For any types A, B in GTT, if for every interpretation $(\!|\cdot|\!) : \Sigma \to \mathbb{C}$, $[\![A]\!]_0 \sqsubseteq [\![B]\!]_0$ holds, then $A \sqsubseteq A'$ is provable in GTT.*
(2) *For any GTT contexts $\Phi : \Gamma \sqsubseteq \Gamma'$, types $A \sqsubseteq A'$, and terms $\Gamma \vdash t : A$ and $\Gamma' \vdash t' : A'$ generated from Σ, if for every interpretation $[\![t]\!]_0 \sqsubseteq [\![t']\!]_0$, then $\Phi \vdash t \sqsubseteq t' : A \sqsubseteq A'$ is provable in GTT.*

Proof. As with the proof of Theorem 4.11, we show that the interpretation used in the soundness Theorem 4.32 constructs a GTT category, i.e. $A \sqsubseteq A'$ holds precisely when $A \sqsubseteq A'$ is derivable and similarly for $t \sqsubseteq t'$. The proof is routine as the rules correspond precisely to the semantic notions. □

Together these theorems imply that the syntax is initial: the semantics given by Definition 4.32 is the unique extension making a morphism of GTT categories using the GTT category structure of Theorem 4.33. To formalize this, we define first a category of GTT signatures GttS in analogy with the category of PTT signatures PttS. We define the category of GTT categories to have as functors CPM functors that preserve the GTT structure.

Definition 4.34 (Category of GTT Categories)**.** We define the category GttC to have as objects GTT categories and as morphisms $F : \mathbb{C} \to \mathbb{D}$ functors F that preserve all GTT structure on the nose in that it
(1) preserves greatest objects $F(\top) = \top$
(2) preserves local bottoms $F(\bot) = \bot$
(3) preserves exponentials in that $F(A \to B) = FA \to FB$ and $F(\lambda(f)) = \lambda(Ff)$ and $F(\mathrm{app}) = \mathrm{app}$.
(4) preserves cartesian products in that $F(A \times B) = FA \times FB$ and $F(\mathrm{pair}(f, g)) = \mathrm{pair}(F(f), F(g))$ and $F(\pi_i) = \pi_i$ and $F(1) = 1$ and $F(\mathrm{unit}) = \mathrm{unit}$

Note that we need not specify that the functors preserve the upcasts/downcasts because any functor of CPMs automatically preserves upcasts/downcasts up to $\sqsupseteq\sqsubseteq$, as shown for double categories in [32].

Lemma 4.35 (CPM Functors preserve Upcasts/Downcasts)**.** *If $F : \mathbb{C} \to \mathbb{D}$ is a CPM functor, then*
(1) *If u is an upcast for $A \sqsubseteq A'$, then $F(u)$ is an upcast for $F(A) \sqsubseteq F(A')$.*
(2) *If d is a downcast for $A \sqsubseteq A'$, then $F(d)$ is a downcast for $F(A) \sqsubseteq F(A')$.*

Proof. We show the upcast case, the downcast case is dual. We need to show $\mathrm{id}_{F(A)} \sqsubseteq F(u)$ and $F(u) \sqsubseteq \mathrm{id}_{F(A')}$. Since u is an upcast, $\mathrm{id}_A \sqsubseteq u$. By monotonicity of F, $F(\mathrm{id}_A) \sqsubseteq F(u)$. By functoriality of F, $F(\mathrm{id}_A) = \mathrm{id}_{FA}$, so $\mathrm{id}_{FA} \sqsubseteq F(u)$. $F(u) \sqsubseteq \mathrm{id}_{F(A')}$ follows by the same argument. □

Then as before there is an obvious "complete signature functor" $U : \mathrm{GttC} \to \mathrm{GttS}$, and the interpretation function is a left adjoint, with the proof an extension of the PTT case. This is a slight generalization of the usual notion of left adjoint, because the uniqueness is up to order-equivalence on arrows $\sqsupseteq\sqsubseteq$, since we don't require a choice of interpretation of upcasts/downcasts.

Theorem 4.36 (GTT is the Initial GTT Category generated by a Signature)**.** *The GTT syntax $GTT : GttS \to GttC$ is left adjoint to U.*

Proof. Again we use the universal morphism definition of a left adjoint.

(1) As shown in Theorem 4.33, $GTT(\Sigma)$ is a GTT category.
(2) For any signature Σ, there is a "universal interpretation" $\eta_\Sigma : \Sigma \to U(GTT(\Sigma))$ interpreting the axioms/function symbols as "themselves", as in the universal interpretation for PTT.
(3) For every interpretation $(\!|\cdot|\!) : \Sigma \to U\mathbb{C}$, there is a functor of GTT categories $[\![\cdot]\!] : GTT(\Sigma) \to \mathbb{C}$ given by the construction in the soundness Theorem 4.32. It is a morphism of GTT categories because it is compositional and preserves exponentials, products, etc.
(4) Every interpretation $(\!|\cdot|\!)$ factors through the universal interpretation in that $(\!|\cdot|\!) = U([\![\cdot]\!]) \circ \eta_\Sigma$. This follows by the same argument as the PTT case.
(5) $[\![\cdot]\!]$ is the unique such factorization. Suppose $F : PTT(\Sigma) \to \mathbb{C}$ is a GTT functor such that $(\!|\cdot|\!) = UF$ on objects and $(\!|\cdot|\!) \sqsupseteq\sqsubseteq UF$ on morphisms. We show that $F_0 = [\![\cdot]\!]_0$ on objects and $F_1 \sqsupseteq\sqsubseteq [\![\cdot]\!]_2$ on morphisms. First, by induction on types.
   (a) For base types, as before $F_0(X) = (\!|X|\!)_0 = [\![X]\!]_0$ because F factors through $(\!|\cdot|\!)$.
   (b) For the dynamic type, $F_0(?) = \top = [\![?]\!]_0$ because F is a GTT functor.
   (c) For function types $F_0(A \to B) = F_0(A) \to F_0(B)$ because F is a GTT functor, then by induction this is equivalent to $[\![A]\!]_0 \to [\![B]\!]_0 = [\![A \to B]\!]_0$. The product and unit cases are similar.

   Next, by induction on terms.
   (a) For variables and function symbols, the same inductive argument as the PTT case follows.
   (b) For the type error $F_1(\mathcal{O}) = \bot = [\![\bot]\!]_2$ because F is a GTT functor.
   (c) For the upcast, we need to show $F_1(\langle A' \nwarrow A\rangle t) \sqsupseteq\sqsubseteq [\![\langle A' \nwarrow A\rangle t]\!]_2$. By functoriality we know $F_1(\langle A' \nwarrow A\rangle t) = F_1(\langle A' \nwarrow A\rangle x) \circ (F_1(t))$ and by definition of the interpretation function, $[\![\langle A' \nwarrow A\rangle t]\!]_2 = u_{[\![A]\!]_0,[\![A']\!]_0} \circ ([\![t]\!]_2)$ so by induction and monotonicity it is sufficient to show $F_1(\langle A' \nwarrow A\rangle x) \sqsupseteq\sqsubseteq u_{[\![A]\!]_0,[\![A']\!]_0}$. By Lemma 4.17, it is sufficient to show that $F(\langle A' \nwarrow A\rangle x)$ is an upcast, which follows from Lemma 4.35. The downcast case follows by the same argument.
   (d) For $\lambda x.t$, we have $F_1(\lambda x.t) = \lambda(F_1(t))$, and by induction $\lambda(F_1(t)) \sqsupseteq\sqsubseteq \lambda([\![t]\!]_2) = [\![\lambda x.t]\!]_2$. The pair/unit introduction cases is similar.
   (e) For application tu, we calculate

$$\begin{aligned} F_1(tu) &= F_1(x_1 x_2) \circ (F_1(t), F_1(u)) && \text{(functoriality)} \\ &= \mathrm{app} \circ (F_1(t), F_1(u)) && (F \text{ preserves application}) \\ &\sqsupseteq\sqsubseteq \mathrm{app}([\![t]\!]_2, [\![u]\!]_2) && \text{(induction)} \\ &= [\![tu]\!]_2 \end{aligned}$$

   The projection cases are similar. □

5. Semantic Contract Interpretation

As a next step towards constructing specific GTT categories, we define a general *contract construction* that provides a semantic account of the "contract interpretation" of gradual typing, which models a gradual type by a pair of casts. The input to our contract construction is a locally thin 2-category $\mathbb{C}$, whose objects and arrows should be thought of as the types and terms of a programming language, and each hom-set $\mathbb{C}(A, B)$ is ordered by an "approximation ordering", which is used to define term dynamism in our eventual model. We require each hom-set to have a least element (the type error), and the category to be cartesian closed (function and product types/contexts) in the strict sense of a 2-category whose underlying category is cartesian closed and where λ, application, pairing and projection are all functorial in 2-cells. The contract construction then implements gradual typing using the morphisms of the non-gradual "programming language" $\mathbb{C}$.

5.1. **Coreflections.** To build a GTT model from $\mathbb{C}$, we need to choose an interpretation of type dynamism (the ordering on objects of the CPM) that induces appropriate casts, which we know by Theorem 3.6 must be Galois connections that satisfy the retract axiom. Such Galois connections are called Galois insertions (in order theory), coreflections (in category theory) and embedding-projection pairs (in domain theory). In this paper we use the term coreflection since it is shortest. While we presented the retract axiom earlier as an $\sqsupseteq\sqsubseteq$, in all of our models the semantics of the composition $\langle A \swarrow A' \rangle \circ \langle A' \nwarrow A \rangle$ is strictly equal to the identity so we will make a model using "strict" coreflections because it is slightly simpler. Since type dynamism judgments must induce a coreflection, we will construct a model where the semantics of a type dynamism judgment $A \sqsubseteq A'$ *is* a coreflection. However, there can be many different coreflections between two objects of our 2-category $\mathbb{C}$, so this first step of our construction does not produce a preorder category, where type dynamism is an *ordering*, but rather a *double category*. Double categories generalize preorder categories in the same way that categories generalize preorders. Double categories are categories internal to the category of small categories, rather than the category of preorders. The ordering on objects is generalized to proof-relevant data specifying a second class of *vertical morphisms*, and the ordering on terms becomes a notion of 2-dimensional "square" between morphisms.

Definition 5.1 (Double Category)**.** A double category consists of

(1) A category of objects and "vertical" arrows $\mathbb{C}_0$
(2) A category of "horizontal" arrows and 2-cells $\mathbb{C}_1$
(3) source, target and identity *functors* with associativity and unitality axioms.

In the model we build from $\mathbb{C}$, the vertical morphisms will model type dynamism and be coreflections, while the *horizontal* morphisms of a preorder category will be arbitrary morphisms of $\mathbb{C}$ and model terms. We still require only double categories that are *locally thin*, in that there is at most one 2-cell filling in any square. Thus, the first step of our contract construction can be summarized as creating a double category that is an equipment with the retract property, i.e. a double category modeling upcasts and downcasts . We can think of this as a model of a "type dynamism proof-relevant" system where there might be many different ways that $A \sqsubseteq A'$ (the next step in the construction will remedy this). Then we get an interpretation of *term dynamism* $\Phi \vdash t \sqsubseteq t' : A \sqsubseteq A'$ as well, but as squares whose sides are on the *proofs* that $\Gamma \sqsubseteq \Gamma'$ and $A \sqsubseteq A'$ and the terms t and t'. Given specific coreflections $(u_A, d_A) : A \triangleleft A'$ and $(u_B, d_B) : B \triangleleft B'$ in $\mathbb{C}$, then a 2-cell from $f : A \to B$ to

$f' : A' \to B'$ along them should be thought of as a *logical relatedness proof*. For instance, in a set-theoretic model any coreflection induces a *relation* between its domain and codomain, so for instance we have a relation $\sqsubseteq_{A,A'}$ that gives us a notion of when an element of A is less dynamic than an element of A' by $x \sqsubseteq_{A,A'} x'$ if $u_A(x) \sqsubseteq_{A'} x'$ or equivalently $x \sqsubseteq_A d_A(x')$. Then a 2-cell from f to g exists if for every $x \sqsubseteq_{A,A'} x'$ then $f(x) \sqsubseteq_{B,B'} f'(x')$. More formally, we can make the following construction, a slight variation on a construction in [32].

Definition 5.2 (Equipment of Coreflections [32])**.** Given a 2-category $\mathbb{C}$ we construct a (double category) equipment CoReflect($\mathbb{C}$) as follows.

(1) Its object category has $\mathbb{C}_0$ as objects and (strict) coreflections in $\mathbb{C}$ as morphisms, i.e., a vertical morphism $A \triangleleft B$ is an adjoint pair of morphisms $u : A \to B$ and $d : B \to A$ where the unit is an equality: $d \circ u = \mathrm{id}$. Composition of coreflections is covariant in the left adjoint and contravariant in the right adjoint: $(u, d) \circ (u', d') = (u \circ u', d' \circ d)$.
(2) Its arrow category CoReflect($\mathbb{C}$)$_1$ has morphisms of $\mathbb{C}$ as objects and a 2-cell from $f : A \to B$ to $f' : A' \to B'$ is a triple of a coreflection $(u_A, d_A) : A \triangleleft A'$, a coreflection $(u_B, d_B) : B \triangleleft B'$ and a *morphism of coreflections*, i.e., a 2-cell in $\mathbb{C}$ $\alpha : u_B \circ f \Rightarrow f' \circ u_A$ which by a simple calculation can be equivalently presented as a morphism $\alpha' : f \circ d_A \Rightarrow d_B \circ f'$.
(3) The upcast from (c_l, c_r) is c_l and the downcast is c_r.

As is well-known in domain theory, covariant, contravariant and any mixed-variance functor preserves coreflections, [43, 36], so the product and exponential functors of $\mathbb{C}$ extend to be functorial also in vertical arrows. This produces the classic "wrapping" construction familiar from higher-order contracts [11]:

$$(u, d) \to (u', d') = (d \to u', u \to d')$$

This construction preserves the structure from $\mathbb{C}$ that will be needed to make a model of gradual type theory:

Theorem 5.3 (Properties of CoReflect($\mathbb{C}$))**.**

(1) *If $\mathbb{C}$ is locally thin then so is CoReflect($\mathbb{C}$).*
(2) *If a 2-category $\mathbb{C}$ has (pseudo) products and exponentials, then so does CoReflect($\mathbb{C}$) because all functors preserve coreflections.*
(3) *If C has local $\perp$s then so does CoReflect($\mathbb{C}$).*

We conjecture that this construction has a universal property: the coreflection construction should be right adjoint to the forgetful functor to 2-categories from the double category of coreflective equipments. Intuitively this means that this is the "maximal" way to add vertical arrows to a 2-category to make it a coreflective equipment.

5.2. **Vertical Slice Category.** The double category CoReflect($\mathbb{C}$) is not yet a model of gradual type theory for two reasons. First, gradual type theory requires a dynamic type: every type should have a canonical coreflection into a specific type. Second, type dynamism in GTT is *proof-irrelevant*, because the rules do not track different witnesses of $A \sqsubseteq A'$, but there may be different coreflections from A to A'. It turns out that we can solve both problems at once by taking what we call the "vertical slice" category[4] over an object $D \in$ CoReflect($\mathbb{C}$)

[4]This definition of vertical slice category is not *quite* the most natural from a higher categorical perspective because the horizontal arrows ignore the chosen object, but it is more useful for our purposes.

that is rich enough to serve as a model of the dynamic type. In CoReflect($\mathbb{C}$)/D, the objects are not just an object A of $\mathbb{C}$, but an object *with* a vertical morphism into D, in this case a coreflection written $(u_A, d_A) : A \triangleleft D$.[5] Thus, gradual types are modeled as coreflections into the dynamic type, analogous to Scott's "retracts of a universal domain" [31]. Then a vertical arrow from $(u_A, d_A) : A \triangleleft D$ to $(u_B, d_B) : B \triangleleft D$ is a coreflection $(u_{A,B}, d_{A,B}) : A \triangleleft B$ that *factorizes* $u_A = u_B \circ u_{A,B}$ and $d_A = d_{A,B} \circ d_B$: this means the enforcement of A's type can be thought of as also enforcing B's type. Since upcasts are monomorphisms and downcasts are epimorphisms, this factorization is *unique* if it exists, so there is at most one vertical arrow between any two objects of CoReflect($\mathbb{C}$)/D. We could weaken this to monomorphism up to $\sqsupseteq\sqsubseteq$ rather than strict equality, in which case the factorization would only be *essentially unique*, i.e. any two factorizations would be equivalent. Since our models produce this stricter form of monomorphism/epimorphism, we defer exploring a weak variant to future work. Further, the identity coreflection $(\mathrm{id}, \mathrm{id}) : D \triangleleft D$ is a vertically greatest element since any morphism is factorized by the identity.

Definition 5.4 (Vertical Slice Category)**.** Given any double category $\mathbb{E}$ and an object $D \in \mathbb{E}$, we can construct a double category $\mathbb{E}/D$ by defining $(\mathbb{E}/D)_0$ to be the slice category $\mathbb{E}_0/D$, a horizontal morphism from $(c : A \triangleleft D)$ to $(d : B \triangleleft D)$ to be a horizontal morphism from A to B in $\mathbb{E}$, and the 2-cells are similarly inherited from $\mathbb{E}$.

Next consider a cartesian closed structure on CoReflect($\mathbb{C}$)/D. The action of $\to$ (respectively $\times, 1$) on objects is given by composition of the action in CoReflect($\mathbb{C}$) $(u, d) \to (u', d')$ with an *arbitrary choice* of "encoding" of the "most dynamic function type" $(u_\to, d_\to) : (D \to D) \triangleleft D$. In most of the models we consider later, D is a sum and this coreflection simply projects out of the corresponding case, failing otherwise. This reflects the separation of the function contract into "higher-order" checking $(u, d) \to (u', d')$ and "first-order tag" checking $(u_\to, d_\to)$ that has been observed in implementations [14].

We summarize the relevant results in the following theorem:

Theorem 5.5 (Vertical Slice Properties)**.**

(1) *If $\mathbb{C}$ is an equipment, then so is $\mathbb{C}/D$.*
(2) *If $\mathbb{C}$ is cartesian, any pair of vertical morphisms $e_\times : D \times D \triangleleft D$ and $e_1 : 1 \triangleleft D$ give $\mathbb{C}/D$ the structure of a cartesian double category by defining $c \times d$ to be $e_\times \circ (c \times d)$ and inheriting the relevant morphisms from $\mathbb{C}$'s cartesian structure.*
(3) *If $\mathbb{C}$ is closed, any vertical morphism $e_\to : (D \to D) \triangleleft D$ gives $\mathbb{C}/D$ the structure of a closed double category by defining $c \to d = e_\to \circ (c \to d)$.*
(4) *$\mathbb{C}/D$ is vertically thin (i.e., a preorder category) if and only if every vertical morphism in $\mathbb{C}$ is a monomorphism.*
(5) *If $\mathbb{C}$ has local $\bot$s then so does $\mathbb{C}/D$*

5.3. **Summary.** Finally, we construct a multicategory Multi($\mathbb{C}$) from the double category CoReflect($\mathbb{C}$)/D. A multiarrow $A_0, \ldots; B$ is given by a horizontal arrow from $A_1 \times ...1$ to B in CoReflect($\mathbb{C}$)/D. The ordering $A \sqsubseteq A'$ is given by the vertical arrows $A \triangleleft A'$ of CoReflect($\mathbb{C}$)/D (i.e. coreflections), which is lifted pointwise to contexts by the definition of a CPM. The ordering $f : (A_0, \ldots; B) \sqsubseteq g : (A'_0, \ldots; B')$ is given by squares in CoReflect($\mathbb{C}$)/D (using the action/monotonicity of $\times$ on the pointwise orderings $A_i \sqsubseteq A'_i$ of the context).

[5] We do not write $A \sqsubseteq D$ because coreflections are not a preorder.

Definition 5.6 (Cartesian Preorder Multicategory from a Cartesian Preorder Category)**.** If $\mathbb{C}$ is a cartesian preorder category, then we can construct a CPM category Multi($\mathbb{C}$) by
(1) $\text{Multi}(\mathbb{C})_0 = \mathbb{C}_0$
(2) $\text{Multi}(\mathbb{C})(A_0, \ldots; B) = \mathbb{C}_1(A_0 \times \cdots \times 1; B)$

Proof. This follows from a quite general result of [6]. □

Combining these constructions, we produce:

Theorem 5.7 (Contract Model of Gradual Typing)**.** *If $\mathbb{C}$ is a locally thin cartesian closed 2-category with local $\bot$s, then for any object $D \in \mathbb{C}$ with chosen coreflections $c_\to : (D \to D) \lhd D$, $c_\times : (D \times D) \lhd D$, and $c_1 : 1 \lhd D$, then $(\mathit{Multi}(\mathit{CoReflect}(\mathbb{C})/\mathit{id}_d), c_\to, c_\times, c_1)$ is a GTT category.*

6. Concrete Models

Now that we have identified a general method of constructing models of gradual type theory, we can produce some concrete models by producing suitable 2-categories.

6.1. **Pointed Preorder Model.** First, we present a simple first-order preorder model. The model is first-order because it models the fragment of gradual type theory without function types. However, by not accommodating function types it is much more elementary. The 2-category for the preorder model is the category $\mathit{PreOrd}_\bot$ whose objects are preorders with a least element, which following domain-theoretic terminology we call "pointed" preorders, and whose morphisms are monotone functions (that don't necessarily preserve $\bot$) and 2-cells are given by the obvious ordering on morphisms. This is a cartesian locally thin 2-category with local $\bot$s (also closed but we will not use this). To construct a suitable dynamic type, we can start with a base set, such as the natural numbers $\mathbb{N}$ and construct the dynamic type by finding a solution to the equation:

$$D \cong \mathbb{N}_\bot \oplus (D \times D)$$

where $\oplus$ is the wedge sum of pointed preorders that identifies the $\bot$s of the two sides. Since this is a *covariant* domain equation, this can be constructed as a simple colimit, and the solution has as elements finite binary trees whose leaves are either natural numbers or a base element $\bot$. The ordering on the trees $T \sqsubseteq T'$ holds when T can be produced from T' by replacing some number of subtrees by $\bot$, a simple model of the dynamism ordering. Finally to get a model, the upcast of the coreflection $D \times D \lhd D$ simply injects to the right side of $\oplus$ and the downcast errors on the $\mathbb{N}_\bot$ case and otherwise returns the pair.

6.2. **Scott's Model.** Next we present two models based on domains that are operationally *inadequate* because they identify the dynamic type error and diverging programs. The first is merely a new presentation of Dana Scott's classical models of untyped lambda calculus but for a gradually typed language [31]. The second is a variation on that construction where product and function types have overlapping representation, showing that the product and function types cannot be proven disjoint in gradual type theory. Both are based on the 2-category of pointed ω-chain complete partial orders, which we simply call *domains* and continuous functions. By standard domain-theoretic techniques (see [43, 36, 29]) we can construct a suitable dynamic type by solving the recursive domain equation:

$$D \cong \mathbb{N}_\bot \oplus (D \times D) \oplus (D \to D)$$

where $\oplus$ is the wedge sum of domains that identifies their least element. The classical technique for solving this equation naturally produce the required coreflections $(D \times D) \triangleleft D$ and $(D \to D) \triangleleft D$.

Next, to get a model in which product and function types are not disjoint we can construct a dynamic type as a *product* of our connectives rather than a sum:

$$D' \cong \mathbb{N}_\perp \times (D' \times D') \times (D' \to D')$$

This is a kind of "coinductive" dynamic type that can be thought of as somewhat object-oriented: rather than an element of the dynamic type being a tagged value, it is something that responds to a set of messages (given by the projections) and if it "doesn't implement" the message it merely returns $\perp$. Then $\langle (? \times ?) \swarrow ? \rangle \langle ? \nwarrow (? \to ?) \rangle x \not\equiv \mho$ because there are elements of the domain that are non-trivial both in the $D \times D$ position and $D \to D$ position.

6.3. **Resolution: Pointed Domain Preorders.** We can combine the best aspects of the domain and pointed preorder models into a single model of pointed, preorder domains, i.e., domains that in addition to their intrinsic domain ordering that models a "divergence ordering" with diverging programs modeled by the divergence-least element have a second, "error ordering" with a least element $\mho$ that models the dynamic type error. The error ordering needs two properties related to the domain structure. First, the diverging element should be maximal, meaning that it is not considered to be more erroring than anything else, which ensures that when we take the wedge sum, we *only* identify $\perp$s and don't otherwise affect the error ordering. Second, the error ordering should be an *admissible* relation, meaning that a limit of one chain is error-less-than the limit of another, if at every step they are related.

Definition 6.1 (Domain Preorder, Pointed Domain Preorder, Continuous Functions)**.**

(1) A domain preorder X is a set $|X|$ with two orderings $\leq_X$ and $\sqsubseteq_X$ such that $(|X|, \leq_X)$ is an ω-complete pointed partial order with $\leq_X$-least element $\perp_X \in |X|$ and $\sqsubseteq_X$ is a preorder such that
   (a) $\perp_X$ is a $\sqsubseteq_X$-maximal element: if $\perp \sqsubseteq x$ then $x = \perp$
   (b) $\sqsubseteq$ is $\leq$-admissible/closed under limits of $\leq_X$-ω-chains: if $\{x_i\}_{i<\omega}, \{y_i\}_{i<\omega}$ are $\leq_X$-ω-chains in X and $x_i \sqsubseteq y_i$ for every $i < \omega$, then $\bigvee\{x_i\} \sqsubseteq \bigvee\{y_i\}$.
(2) A continuous function of domain preorders is a function of the underlying sets that is continuous with respect to $\leq_X$ and monotone with respect to $\sqsubseteq_X$.
(3) A pointed domain preorder X is a domain preorder with a $\sqsubseteq_X$-least element $\mho \in |X|$. Continuous functions of pointed domain preorders are just continuous functions of the underlying domain preorders

We model our types as pointed domain preorders because they have an interpretation for both divergence $\perp$ and type error $\mho$, and our terms will be modeled as continuous functions. First, we show that both categories are cartesian closed.

Lemma 6.2. *The categories of domain preorders and pointed domain preorders with continuous functions are cartesian closed.*

Proof. Since the category of pointed domain preorders is a full subcategory of domain preorders, we can show that it has unit, product and exponential if the category of domain

preorders has the corresponding universal property and the operations happen to produce pointed domain preorders when given pointed domain preorders.

(1) 1 is the singleton $\{*\}$ with the unique ordering. The unique element is a $\leq$ and $\sqsubseteq$-least element.
(2) $X \times Y$ is constructed as follows. The underlying set is the product of underlying sets $|X \times Y| = |X| \times |Y|$. Each ordering is point-wise, and limits are taken pointwise. Because limits are pointwise, $\sqsubseteq_X \times \sqsubseteq_Y$ is admissible. If X, Y have $\mho$s, then $X \times Y$ has a $\sqsubseteq$-least element $\mho_{X\times Y} = (\mho_X, \mho_Y)$.

   The pairing, and projection functions are clearly continuous in $\leq$ and monotone in $\sqsubseteq$.
(3) $X \to Y$. The underlying set is the set of continuous functions of domain preorders from X to Y. The orderings are given pointwise.

   Since this is a subset of the set of continuous functions of the underlying domains, to show that it is closed under $\leq$-ω-chains, we just need to show that the limit of a chain of monotone functions is monotone. The limit of the chain $\{f_i\}$ is just $\lambda x. \bigvee\{f_i(x)\}$. Given $x \sqsubseteq_X y$, we need to show $\bigvee\{f_i(x)\} \sqsubseteq_Y \bigvee\{f_i(y)\}$. Since $\sqsubseteq_Y$ is admissible, it is sufficient to show that $f_i(x) \sqsubseteq f_i(y)$ for each i, which follows by monotonicity of f_i.

   Next, the pointwise ordering is clearly admissible because limits are taken pointwise.

   Next, if $f : X \times Y \to Z$ is a continuous function of preorder domains, $\lambda_y f : X \to (Y \to Z)$ is clearly continuous. And the application function $app : (X \to Y) \times X \to Y$ is also continuous.

   If Y has a $\mho$, then $X \to Y$ has a least element $\mho_{X\to Y} = \lambda x.\mho_Y$. □

Next, we will solve the recursive domain equations using the classic construction using ep pairs. Our category of domain preorders is an O-category in the sense of [43, 36], but does not have all ω^{op} limits due to the restriction that the $\bot$ element be $\sqsubseteq$-maximal. However, if the ω^{op} diagram is made of strict ($\bot$-preserving) functions, the limit does exist, and this is all the central theorem of [36] actually requires because projections are always strict.

Lemma 6.3. *The category of domain preorders and continuous functions is an O-category with the point-wise $\leq$ ordering.*

Proof. We need to show two properties

(1) First, every hom set is an ω-cppo as shown in lemma 6.2.
(2) Next we need to show composition is $\leq$-continuous. Since composition can be defined as $\lambda f.\lambda g.\lambda x.g(f(x))$ it is sufficient to show that $\lambda : (X \times Y \to Z) \to (X \to (Y \to X))$ is continuous (application is continuous because the category is cartesian closed). Given a chain of $f_i : X \times Y \to Z$, on the one hand

$$(\bigvee_i \{\lambda f_i\})(x)(y) = \bigvee_i \{f_i(x, y)\}$$

   and on the other hand

$$(\lambda \bigvee_i \{f_i\})(x)(y) = (\bigvee_i \{f_i\})(x, y) = \bigvee_i \{f_i(x, y)\}$$ □

Lemma 6.4. *The category of domain preorders has all ω^{op} limits where the diagram is made of $\bot$-preserving maps.*

Proof. Given an ω^{op} diagram of domain preorders $f_i : D_{i+1} \to D_i$ where $i \in \omega$, the proposed limit D_ω has as underlying set

$$|D_\omega| = \{d : \Pi_{i\in\omega} D_i \mid \forall i \in \omega.\ f_i(d_{i+1}) = d_i\}$$

The orderings $\leq, \sqsubseteq$ are both given point-wise, and admissibility follows from that. $\lambda i.\bot_{D_i} \in D_\omega$ because $f_i(\bot_{i+1}) = \bot_i$ by strictness. Since the order is pointwise, it is a least element and maximal. Finally, because the ordering is pointwise, a function into D_ω is continuous if and only if its composition with each projection is continuous, so D_ω is the limit of the given diagram. □

Lemma 6.5. *The product $\times$, exponential $\to$, wedge sum $\oplus$ and adjoining an error $\cdot_\mho$ are all locally continuous mixed-variance functors on the category of domain preorders.*

Proof. (1) $\times$: $f \times g = \lambda(x, y).(fx, gy)$, this is obviously functorial and is locally continuous because the orderings are point-wise.

(2) $\to$: $f \to g = \lambda h.g \circ h \circ f$, obviously functorial and locally continuous because the ordering is pointwise.

(3) $X_\mho$ The underlying set is $\{0\} \times |X| \uplus \{1\} \times \{\mho\}$ and the orderings are defined as

$$(i, z) \leq (i', z') = i = i' \wedge z \leq z'$$

and

$$(i, z) \sqsubseteq (i', z') = i = 1 \vee (i = i' = 0 \wedge z \sqsubseteq_X z')$$

Which is clearly a domain preorder. The functorial action is defined by

$$f_\mho(1, \mho) = \mho$$
$$f_\mho(0, x) = (0, f(x))$$

which is clearly functorial and continuous.

(4) $\oplus$. $|X \oplus Y| = (\{0\} \times X \uplus \{1\} \times Y)/((0, \bot_X) = (1, \bot_Y))$ with the $\leq$-ordering defined as

$$(i, z) \leq_{X \oplus Y} (i', z') = (z = \bot_X) \vee z = \bot_Y \vee (i = i' \wedge z \leq z')$$

And error ordering similarly defined as

$$(i, z) \sqsubseteq_{X \oplus Y} (i', z') = (z, z' \in \{\bot_X, \bot_Y\}) \vee (i = i' \wedge z \sqsubseteq z')$$

These define a domain and preorder structure respectively, and $\sqsubseteq$ is admissible. Since $\bot_X, \bot_Y$ are maximal, $\bot_{X \oplus Y}$ is also maximal. □

We can then construct a suitable dynamic type using the construction of [36].

Theorem 6.6. *There exists a domain preorder with an isomorphism*

$$i : D \cong (\mathbb{N}_\bot)_\mho \oplus (D_\mho \times D_\mho) \oplus (D_\mho \to D_\mho)$$

Then we interpret the dynamic type as $D_\mho$. The coreflection $(e_1, p_1) : 1 \lhd D_\mho$ is the unique $\mho$-coreflection: $e_1(*) = \mho$ and $p_1(x) = *$. The coreflection $(e_\times, p_\times) : D_\mho \times D_\mho \lhd D_\mho$ is defined as

$$e_\times(x) = i((1, x))$$
$$p_\times(\mho) = \mho$$
$$p_\times(\bot) = \bot$$
$$p_\times(0, n : (\mathbb{N}_\bot)_\mho) = \mho$$
$$p_\times(1, t : (D_\mho \times D_\mho)) = t$$
$$p_\times(2, f : (D_\mho \to D_\mho)) = \mho$$

Which is monotone because $\bot$ is a maximal element. The coreflection $(e_{\to}, p_{\to}) : D_{\mathcal{U}} \to D_{\mathcal{U}} \triangleleft D_{\mathcal{U}}$ is defined in the same way, but using the function case of the sum.

7. Related and Future Work

Since the original conference publication of this article [24], we have developed [25] a version of gradual type theory based on call-by-push-value, which extends call-by-value and call-by-name, and generalizes the type theory we have presented here. The implementations of the dynamic value type and dynamic computation type in [25] are based on the two models in Section 6.2. We also developed operational models of CBPV gradual type theory, based on an interpretation of term dynamism as a kind of contextual approximation, following [23]. While we did not develop a categorical semantics for CBPV gradual type theory, there should be a similar categorical semantics to the one presented here, by generalizing from a preorder category to a (strong) adjunction of preorder categories. In particular, our presentation of the operational models there is a refinement of the contract construction we give here, and so should in principle be also described by taking a kind of double category of coreflections and vertical slice.

7.1. **Logic and Semantics of Dynamism.** Our logic and semantics of type and term dynamism builds on the formulation introduced with the gradual guarantee in [33], but the rules of our system differ in two key ways. First, our system includes the β, η equivalences as equi-dynamism axioms, making term dynamism a more semantic notion. Second, we only allow casts that are either upcasts or downcasts (as defined by type dynamism), whereas their system allows for a more liberal "compatibility" condition. Accordingly our rules of dynamism for casts are slightly different, but where it makes sense, the rules of the two systems are interderivable. Modifying their cast rule to our syntax and ignoring any compatibility constraint on the casted types, they have the following two rules:

$$\frac{\Phi \vdash t_1 \sqsubseteq t_2 : A_1 \sqsubseteq A_2 \qquad (A_1 \sqsubseteq B_2)}{\Phi \vdash t_1 \sqsubseteq \langle B_2 \Leftarrow A_2 \rangle t_2 : A_1 \sqsubseteq B_2} \textsc{Cast-R}$$

$$\frac{\Phi \vdash t_1 \sqsubseteq t_2 : A_1 \sqsubseteq A_2 \qquad (B_1 \sqsubseteq A_2)}{\Phi \vdash \langle B_1 \Leftarrow A_1 \rangle t_1 \sqsubseteq t_2 : B_1 \sqsubseteq A_2} \textsc{Cast-L}$$

Then we see that our four rules for upcast and downcasts are the special case where the casts involved are upcasts or downcasts. In the reverse direction, if we *define* the "oblique" casts as $\langle B \Leftarrow A \rangle t = \langle B \swarrow ? \rangle \langle ? \nwarrow A \rangle t$, we can derive their rules in 8. First, their cast right rule follows easily by applying our sequent-style cast rules DR(S) and UR(S). The left rule takes slightly more work, using the retract axiom, because we can't cast up to ? on the left side because it might be more dynamic than A_2. Instead, we first prove that we can define the oblique cast $\langle B \Leftarrow A \rangle t$ not just as the cast through ?, but also through *any* C with $A, B \sqsubseteq C$ using the retract axiom and composition of upcasts, downcasts. Then we pick C in the Cast-L case to be A_2, and then proof proceeds dually to the Cast-R case.

As a relational logic with a sound and complete categorical semantics, it has commonalities with logics for parametric polymorphism [30], and the categorical semantics in terms of *reflexive graph categories* which are like double categories where vertical arrows lack composition [26]. In particular the System P logic presented in [8] is similar to a "dynamism proof-relevant" version of preorder type theory. Additionally, the bifibration condition of

$$\dfrac{\dfrac{\Phi \vdash t_1 \sqsubseteq t_2 : A_1 \sqsubseteq A_2}{\Phi \vdash t_1 \sqsubseteq \langle ? \nwarrow A_2 \rangle t_2 : A_1 \sqsubseteq ?}\ \mathrm{UR(S)}}{\Phi \vdash t_1 \sqsubseteq \langle B_2 \swarrow ? \rangle \langle ? \nwarrow A_2 \rangle t_2 : A_1 \sqsubseteq B_2}\ \mathrm{DR(S)}$$

$$\text{if } A, B \sqsubseteq C$$

$$\langle B \swarrow ? \rangle \langle ? \nwarrow A \rangle t \sqsupseteq\sqsubseteq \langle B \swarrow C \rangle \langle C \swarrow ? \rangle \langle ? \nwarrow C \rangle \langle C \nwarrow A \rangle t \sqsupseteq\sqsubseteq \langle B \swarrow C \rangle \langle C \nwarrow A \rangle t$$

$$\dfrac{\dfrac{\Phi \vdash t_1 \sqsubseteq t_2 : A_1 \sqsubseteq A_2}{\Phi \vdash \langle A_2 \nwarrow A_1 \rangle t_1 \sqsubseteq t_2 : A_2 \sqsubseteq A_2}\ \mathrm{UL(S)}}{\Phi \vdash \langle B_1 \swarrow A_2 \rangle \langle A_2 \nwarrow A_1 \rangle t_1 \sqsubseteq t_2 : B_1 \sqsubseteq A_2}\ \mathrm{DL(S)}$$

Figure 8: [33] Cast Rules Derived

[13] is essentially the same as the definition of an *equipment*, but with a twist: in gradual typing every contract induces an adjoint pair of terms, but there every term induces an adjoint pair of *relations*: the graph and "cograph". Hopefully the similarity with parametric logics will be useful in studying the combination of graduality with parametricity.

7.2. **Contracts as Coreflections.** Our semantic model of contracts as coreflections has precedent in much previous work, though we are the first to make precise the relationship to gradual typing's notions of type and term dynamism.

First, Dana Scott's seminal denotational work on models of the lambda calculus is very similar to our vertical slice category: types are modeled as retracts (or their associated idempotent) of a fixed universal domain and morphisms are continuous functions of the underlying domain (ignoring the universal domain). Our treatment of type and term dynamism utilizes additional details of this model, and the move from retracts to coreflections allows us to give our specification for upcasts and downcasts. Additionally, Scott's paper and later denotational work use coreflections to solve mixed-variance domain equations [31, 43, 36]. The key reason is that one cannot construct a solution to $D \cong D \to D$ as a limit or colimit because $\to$ has contravariant and covariant arguments. Instead, one moves to the category of coreflections where $\to$ is covariant in both arguments. Our coreflection model shows that this "trick" is also the reason that the function type constructor is *monotone* with respect to type dynamism. The double category setting allows us to better understand the intertwined relationship between the categories of continuous maps and coreflections and in this respect has much similarity to [29]'s work, much of which could be fruitfully reframed in a double categorical setting.

Henglein's work [14] on dynamic typing defines casts that are retracts to the dynamic type, introduced the upcast-followed-by-downcast factorization that we use here, and defines a syntactic rewriting relation similar to our term dynamism rules. Further they define a "subtyping" relation that is the same as type dynamism and characterize it by a semantic property analogous to the semantics of type dynamism in our contract model.

Findler and Blume's work on contracts as *pairs of projections* [10] is also similar. There a contract is defined in an untyped language to be given by a *pair* of functions that divide enforcement of a type between a "positive" component that checks the term and a "negative"

component that checks the continuation, naturally supporting a definition of *blame* when a contract is violated. We give no formal treatment of blame in this paper, but our separation into upcasts and downcasts naturally supports a definition of blame analogous to theirs. In their paper, each component c is idempotent and satisfies $c \sqsubseteq \mathrm{id}$. Their work is fundamentally untyped so a direct comparison is difficult. Their pairs of projections are not coreflections between the untyped domain and itself and it doesn't make sense to ask whether our upcasts and downcasts are error projections because they are not endomorphisms. We can say that on the one hand any coreflection with components $u, d : A \lhd ?$ produces an error projection $u \circ d$ on $?$, but then we are left with a single projection rather than two. We might be able to make a more direct comparison using a *semantic* type system over an untyped language, in the style of [5].

Recent work on interoperability in a (non-gradual) dependently typed language [7] defines several variations of Galois connections to serve as models of casts with different properties. This work validates their comments that ordinary monotone Galois connections serve as a model of the upcasts and downcasts associated to type dynamism.

7.3. **Frameworks for Gradual Typing.** There are two recent proposals for a more general theory of gradual typing: Abstracting Gradual Typing (AGT) [12] and the Gradualizer [4]. Broadly, their systems and ours are similar in that type dynamism and graduality are central and a gradually typed language is constructed from a statically typed language. Gradual type theory is quite different in that it is based on an axiomatic semantics, whereas both of theirs are based on operational semantics. As such our notion of gradual type soundness is stronger than theirs: we assert program equivalences whereas their soundness theorem is related to the syntactic type soundness theorem of the static language. Their systems also develop a *surface syntax* for gradually typed languages (including implicit casts and gradual type checking), whereas our logic here only applies to the *runtime semantics* of the language. In particular, their languages have *implicit* casts which are elaborated into an explicit cast calculus that is more similar to our type theory. Their approaches also consider the problem of how a gradual type *checker* should balance the demands of disallowing terms that will produce type errors with the requirement that the language still have a subset that supports a dynamically typed programming style.

The AGT framework also allows for a variation on gradual typing where only some types are "gradual" in the sense that they are less dynamic than the dynamic type. For instance, by removing the rule $A \to B \sqsubseteq ?$ we get a dynamic type that only embeds first-order types, and so rules out costly higher-order casts. We can accommodate this in our axiomatics by simply limiting the $A \sqsubseteq ?$ rule to only apply to certain types A, the definition of the casts can remain the same. In fact our first-order model or pointed preorders in Section 6.1 would be a model of such a system since the category that interprets terms is cartesian closed. Finally, AGT is based on abstract interpretation and uses a Galois connection between gradual types and sets of static types that is actually a coreflection itself, but we do not see a precise relationship to our use of coreflections, and so this may just reflect the ubiquity of this mathematical concept.

7.4. **Cast Factorization.** The factorization of an arbitrary cast $A \Rightarrow B$ into an upcast to $?$ followed by a downcast is superficially similar to the work of [35], which collapse a sequence of casts starting at A and ending at B into a downcast to $A \sqcap B$ followed by an

upcast to B. Note that their factorization is in fact opposite: ours is an upcast followed by a downcast. The factorization we present is trivial and was originally presented in [14], whereas theirs involves some actual computation of a type and is similar to *image factorization*. Furthermore, it was shown in [12] that the correctness of factorization through $A \sqcap B$ is not always possible and is highly dependent on the available language of gradual types, whereas our factorization solely depends on the presence of a dynamic type, which could even be weakened to the two types having a common $\sqsubseteq$-supertype.

Relative to this related work, we believe the axiomatic specification of casts via a universal property relative to dynamism is a new idea in gradual typing, as is our categorical semantics and the presentation of the contract interpretation as a model construction.

7.5. **Future Work.** The clearest challenges for future work are the axiomatization of gradual typing with more advanced typing features. For instance, the combination of gradual typing and parametric polymorphism has proven quite complex [21, 2, 16, 40]. If we could show that the combination of graduality with parametricity has a unique implementation, as we have shown here for simple typing, it would provide a strong semantic justification for a design.

Additionally, the combination of dependent typing with gradual typing is worth exploring, especially because while dependent contract checking been used for some time and was explicitly inspired in part by dependent typing, no semantic connection has been established between dependent contracts and category-theoretic models of dependent typing. The main difficulty will be the combination of dependent typing with effects, but there has been much recent work in this area [1, 41, 27].

Another interesting application would be to apply our semantics to other forms of gradualization, such as effect typing [3], security typing [39] and refinement typing [18].

We conjecture that much of our semantics will hold over, but with the dynamism and casts being in a different (preorder) category. For instance, effect types can be interpreted as monads and a cast might be interpreted as an embedding-projection pair of morphisms of monads.

References

[1] Daniel Ahman, Neil Ghani, and Gordon D. Plotkin. Dependent types and fibred computational effects. In *International Conference on Foundations of Software Science and Computation Structures (FoSSaCS)*, 2016.

[2] Amal Ahmed, Dustin Jamner, Jeremy G. Siek, and Philip Wadler. Theorems for free for free: Parametricity, with and without types. In *International Conference on Functional Programming (ICFP)*, 2017.

[3] Felipe Bañados Schwerter, Ronald Garcia, and Éric Tanter. A theory of gradual effect systems. In *Proceedings of the 19th ACM SIGPLAN International Conference on Functional Programming*, ICFP '14, pages 283–295, 2014.

[4] Matteo Cimini and Jeremy G. Siek. Automatically generating the dynamic semantics of gradually typed languages. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages*, POPL 2017, pages 789–803, 2017.

[5] Robert L. Constable, Stuart F. Allen, H. M. Bromley, W. R. Cleaveland, J. F. Cremer, R. W. Harper, Douglas J. Howe, T. B. Knoblock, N. P. Mendler, P. Panangaden, James T. Sasaki, and Scott F. Smith. *Implementing Mathematics with the NuPRL Proof Development System*. Prentice Hall, 1986.

[6] G. S. H. Cruttwell and Michael A. Shulman. A unified framework for generalized multicategories. *Theory and Applications of Categories*, 24(21), 2009. `arXiv:arXiv:0907.2460`.

[7] Pierre-Evariste Dagand, Nicolas Tabareau, and Éric Tanter. Foundations of Dependent Interoperability. working paper or preprint, 2017. URL: `https://hal.inria.fr/hal-01629909`.

[8] Brian Patrick Dunphy. *Parametricity As a Notion of Uniformity in Reflexive Graphs*. PhD thesis, University of Illinois at Urbana-Champaign, Champaign, IL, USA, 2002.

[9] Matthias Felleisen. On the expressive power of programming languages. *ESOP'90*, 1990.

[10] Robby Findler and Matthias Blume. Contracts as pairs of projections. In *International Symposium on Functional and Logic Programming (FLOPS)*, April 2006.

[11] Robert Bruce Findler and Matthias Felleisen. Contracts for higher-order functions. In *International Conference on Functional Programming (ICFP)*, pages 48–59, September 2002.

[12] Ronald Garcia, Alison M. Clark, and Éric Tanter. Abstracting gradual typing. In *ACM Symposium on Principles of Programming Languages (POPL)*, 2016.

[13] Neil Ghani, Patricia Johann, Fredrik Nordvall Forsberg, Federico Orsanigo, and Tim Revell. Bifibrational functorial semantics for parametric polymorphism. In *Proceedings of Mathematical Foundations of Program Semantics*, 2015.

[14] Fritz Henglein. Dynamic typing: Syntax and proof theory. *Science of Computer Programming*, 22(3):197–230, 1994.

[15] Atsushi Igarashi, Peter Thiemann, Vasco Vasconcelos, and Philip Wadler. Gradual session types. In *International Conference on Functional Programming (ICFP)*, 2017.

[16] Yuu Igarashi, Taro Sekiyama, and Atsushi Igarashi. On polymorphic gradual typing. In *International Conference on Functional Programming (ICFP), Oxford, United Kingdom*, 2017.

[17] J. Lambek and P. J. Scott. *Introduction to Higher Order Categorical Logic*. Cambridge University Press, 1986.

[18] Nico Lehmann and Éric Tanter. Gradual refinement types. In *ACM Symposium on Principles of Programming Languages (POPL)*, 2017.

[19] Saunders MacLane. *Categories for the Working Mathematician*. Springer-Verlag, New York, 1971. Graduate Texts in Mathematics, Vol. 5.

[20] Per Martin-Löf. On the meanings of the logical constants and the justifications of the logical laws (sienna lectures). *Nordic Journal of Philosophical Logic*, 1(1):11–60, 1983/1996.

[21] Jacob Matthews and Amal Ahmed. Parametric polymorphism through run-time sealing, or, theorems for low, low prices! In *European Symposium on Programming (ESOP)*, March 2008.

[22] Eugenio Moggi. Notions of computation and monads. *Inform. And Computation*, 93(1), 1991.

[23] Max S. New and Amal Ahmed. Graduality from embedding-projection pairs. In *International Conference on Functional Programming (ICFP), St. Louis, Missouri*, 2018.

[24] Max S. New and Daniel R. Licata. Call-by-name gradual type theory. In *Formal Structures for Computation and Deduction, Oxford England*, 2018.

[25] Max S. New, Daniel R. Licata, and Amal Ahmed. Gradual type theory. In *ACM Symposium on Principles of Programming Languages (POPL), Cascais, Portugal*, 2019.

[26] Peter W. O'Hearn and Robert D. Tennent. Parametricity and local variables. *Journal of the ACM*, 42(3):658–709, May 1995.

[27] Pierre-Marie Pédrot and Nicolas Tabareau. An effectful way to eliminate addiction to dependence. In *IEEE Symposium on Logic in Computer Science (LICS), Reykjavik, Iceland*, 2017.

[28] Frank Pfenning and Rowan Davies. A judgmental reconstruction of modal logic. *Mathematical Structures in Computer Science*, 11:511–540, 2001.

[29] Andrew M. Pitts. Relational properties of domains. *Information and Computation*, 127(2):66 – 90, 1996.

[30] Gordon Plotkin and Martín Abadi. A logic for parametric polymorphism. *Typed Lambda Calculi and Applications*, pages 361–375, 1993.

[31] Dana Scott. Data types as lattices. *Siam Journal on computing*, 5(3):522–587, 1976.

[32] Michael Shulman. Framed bicategories and monoidal fibrations. *Theory and Applications of Categories*, 20(18):650–738, 2008.

[33] Jeremy Siek, Micahel Vitousek, Matteo Cimini, and John Tang Boyland. Refined criteria for gradual typing. In *1st Summit on Advances in Programming Languages*, SNAPL 2015, 2015.

[34] Jeremy G. Siek and Walid Taha. Gradual typing for functional languages. In *Scheme and Functional Programming Workshop (Scheme)*, pages 81–92, September 2006.

[35] Jeremy G. Siek and Philip Wadler. Threesomes, with and without blame. In *ACM Symposium on Principles of Programming Languages (POPL)*, pages 365–376, 2010.

[36] Michael B Smyth and Gordon D Plotkin. The category-theoretic solution of recursive domain equations. *SIAM Journal on Computing*, 11(4), 1982.

[37] Sam Tobin-Hochstadt and Matthias Felleisen. Interlanguage migration: From scripts to programs. In *Dynamic Languages Symposium (DLS)*, pages 964–974, October 2006.

[38] Sam Tobin-Hochstadt and Matthias Felleisen. The design and implementation of typed scheme. In *ACM Symposium on Principles of Programming Languages (POPL), San Francisco, California*, 2008.

[39] Matías Toro, Ronald Garcia, and Éric Tanter. Type-driven gradual security with references. *ACM Transactions on Programming Languages and Systems*, 40(4), December 2018. URL: `http://doi.acm.org/10.1145/3229061`.

[40] Matías Toro, Elizabeth Labrada, and Éric Tanter. Gradual parametricity revisited. In *ACM Symposium on Principles of Programming Languages (POPL), Cascais, Portugal*, 2019.

[41] Matthijs Vákár. In search of effectful dependent types. *CoRR*, 2017. URL: `http://arxiv.org/abs/1706.07997`.

[42] Philip Wadler and Robert Bruce Findler. Well-typed programs can't be blamed. In *European Symposium on Programming (ESOP)*, pages 1–16, March 2009.

[43] Mitchell Wand. Fixed-point constructions in order-enriched categories. *Theoretical Computer Science*, 8(1):13 – 30, 1979.

[44] Roger Wolff, Ronald Garcia, Éric Tanter, and Jonathan Aldrich. Gradual typestate. In *Proceedings of the 25th European Conference on Object-oriented Programming*, ECOOP'11, 2011.