Call-by-Name Gradual Type Theory (Extended Version)

Max S. New*[1] and Daniel R. Licata†[2]

[1]Northeastern University, Boston, USA, maxnew@ccs.neu.edu
[2]Wesleyan University, Middletown, USA, dlicata@wesleyan.edu

February 2, 2018

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*, which were developed in blame calculi and to state 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, which is a semantic analogue of Findler and Felleisen's definitions of contracts, and use it to build some concrete domain-theoretic models of gradual typing.

*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.

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 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 [33, 29] to numerous language features such as parametric polymorphism [1, 16], effect tracking [2], typestate [37], session types [15], and refinement types [18]. Almost all work on gradual typing is based solely on operational semantics, and recent work such as [28] 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.

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 effects. While this has previously been defined using operational semantics and a notion of *blame* [35], 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* [28], 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 a novel *uniform specification* (universal property) for the dynamic type, type errors, and the runtime type casts of a gradually typed language. For soundness, we assert β and η principles as axioms of term dynamism, so it can also be used to reason about programs' 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 [30]) 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 leave call-by-value gradual type theory to future work.

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 [34] 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 from 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* [21]: 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.

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 [28] 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 [28] makes use of a *syntactically less dynamic* 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 [28] this was called "precision"). For example, if $add1 : ? \to \mathbb{N}$ and $\mathsf{true} : ?$, then $add1((? \Leftarrow \mathbb{N})(\mathbb{N} \Leftarrow ?)\mathsf{true})$ (cast true from dynamic to $\mathbb{N}$ and back, to assert it is a number) is syntactically less dynamic than $add1((? \Leftarrow ?)(? \Leftarrow ?)\mathsf{true})$ (where both casts are the identity). Then the gradual guarantee [28] 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 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 $add1$ uses its argument as a number. In contrast, a program that returns a different value than $add1(\mathsf{true})$ 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 (doesn't equate operationally distinguishable terms), then this implies the gradual guarantee [28]. Therefore, we say a model "satisfies graduality" in the same sense that a language satisfies parametricity.

Next, we discuss the relationship between term dynamism and casts/contracts, one of the most novel parts 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 (which, e.g. check the inputs and outputs of functions, check the components of pairs, etc.) . 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 [35]). In gradual type theory, the upcast $\langle B \leftharpoonup A \rangle$ from A to B and the downcast $\langle A \leftharpoondown B \rangle$ 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 $\langle B \twoheadleftarrow ?\rangle\langle ? \leftharpoondown A\rangle t$).

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 $\langle B \leftharpoondown A\rangle t : B$ is the *least* dynamic term of type B that is more dynamic than t. In order-theoretic terms, $\langle B \leftharpoondown A\rangle t : B$ 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 $\langle B \leftharpoondown A\rangle t$ 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 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 $\langle ? \to ? \twoheadleftarrow ?\rangle\langle ? \leftharpoondown ? \times ?\rangle t \mapsto \mho$), which are independent, and can be added as axioms.

Another major contribution of our paper is a soundness and completeness theorem for gradual type theory with respect to semantics in *preorder categories*, i.e., 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 preorder category is 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. 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 sort of preorder category is called an *equipment* and we can use existing constructions and results from that work. In fact, in constructing models we at times need a generalization of preorder categories called *double categories*, which have the same relation to preorder categories that categories have to preorders.

In addition to providing a different perspective on the structure of type and term dynamism, the preorder category semantics of gradual typing enables us to systematically build models of gradual typing. In particular, we cast the "contract interpretation" of casts as a semantic construction of a model of gradual typing from a 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" preorder category whose objects are vertical arrows into the chosen dynamic type. We then instantiate this construction with multiple domain theoretic models. First we show that Dana Scott's classical construction of a model of types from retracts of a universal domain is an instance, but is inadequate for interpreting gradual typing because it conflates type errors and nontermination. Then 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.

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 and casts are defined by rules using the *judgmental structure* of the type theory, which extends the usual judgmental structure of call-by-name typed lambda calculus with a syntax for type and term dynamism. Since the judgmental structure is as important as these types, we present a bare *preorder type theory* (PTT) with no types first. Then we can modularly define what it means for this theory to have a dynamic type, casts, functions and products, and gradual type theory is preorder type theory with all of these.

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

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

Figure 1: Judgment Presuppositions of Preorder Type Theory

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

$$\frac{f \in \Sigma_2(A_1, \ldots, A_n; B) \qquad \Delta \vdash \gamma : x_1 : A_1, \ldots, x_n : A_n}{\Delta \vdash f(\gamma(x_1), \ldots, \gamma(x_n)) : B} \qquad \Gamma \vdash \gamma : \Delta = \gamma \in \prod_{x:A \in \Delta} (\Gamma \vdash A)$$

Figure 2: Preorder Type Theory: Type and Term Structure

hold) are presented in Figure 1, where $A\,\text{type}$ and $\Gamma\,\text{context}$ have no conditions. The types, contexts and terms (Figure 2) are structured as a standard call-by-name type theory. Terms are treated as intrinsically typed with 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 (whose rule we omit). These are all given by a *signature* Σ, formally defined below in 1. A substitution $\Gamma \vdash \Delta$ is defined as usual as giving, for every typed variable in the output context, a term of that type relative to the input context. 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. A type dynamism judgment (Figure 3) $A \sqsubseteq B$ relates two well-formed types, and is read as "A is less dynamic than B". In preorder type theory, the only rules are reflexivity and transitivity, making type dynamism a preorder, and axioms from a signature.

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 $x_1 \sqsubseteq x_1' : A_1 \sqsubseteq A_1', \ldots$ then we have that $t \sqsubseteq t' : B \sqsubseteq B'$.

$$\frac{}{A \sqsubseteq A} \qquad \frac{A \sqsubseteq B \qquad B \sqsubseteq C}{A \sqsubseteq C} \qquad \frac{(A, B) \in \Sigma_1}{A \sqsubseteq B} \qquad \frac{}{\cdot : \cdot \sqsubseteq \cdot} \qquad \frac{\Phi : \Gamma \sqsubseteq \Gamma' \qquad A \sqsubseteq A'}{(\Phi, x \sqsubseteq x' : A \sqsubseteq A') : \Gamma, x : A \sqsubseteq \Gamma', x' : A'}$$

$$\text{when } \Phi : \Gamma \sqsubseteq \Gamma' \text{ and } \Psi : \Delta \sqsubseteq \Delta', \delta : \Gamma \vdash \Delta \text{ and } \delta' : \Gamma' \vdash \Delta'$$

$$\Phi \vdash \delta \sqsubseteq \delta' : \Psi = \forall (x \sqsubseteq x' : A \sqsubseteq A' \in \Psi).\ \Phi \vdash \delta(x) \sqsubseteq \delta'(x') : A \sqsubseteq A'$$

Figure 3: Type and Context Dynamism

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

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

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

Figure 4: Primitive Rules of Term Dynamism

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 arbitrary term dynamism axioms (TmPrec-Ax). First, there is a rule (TmPrec-Var) that relates variables. Next there is a *compositionality* rule (TmPrec-Comp) that allows us to prove dynamism judgments by breaking terms down into components. We elide the definition of *substitution dynamism* $\Phi \vdash \gamma \sqsubseteq \gamma' : \Psi$, which is pointwise term dynamism. Last, we add an appropriate form of reflexivity (TmPrec-Refl) and transitivity (TmPrec-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 "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' must *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 $\langle A' \leftharpoonup A \rangle u$ or $\langle A' \leftharpoondown A'' \rangle u''$).

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 $\Phi \vdash t \sqsubseteq t' : A \sqsubseteq A$.

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 [21]: since it is a general theory of monads, it is not provable that an effect is commutative, but we can add a commutativity axiom and prove additional consequences. Similarly, in our type theory it is not provable without adding non-type-theoretic 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 1 (PTT Signature)**.** *A preorder type theory signature (PTT signature) consists of*

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* $0, 1$*-PTT signatures* Σ_0, Σ_1 *is a set* Σ_2 *with functions* $s : \Sigma_2 \to PTT_0(\Sigma_0)^*$ *and* $t : \Sigma_2 \to PTT_0(\Sigma_0)$*, and whose elements are called* function symbols*.*
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 by those signatures.*
6. *A* 3*-PTT Signature* Σ_3 *relative to* $0, 1, 2$*-signatures* $\Sigma_0, \Sigma_1, \Sigma_2$ *is a subset of* $PTT_1(\Sigma_0, \Sigma_1, \Sigma_2)^2$ *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.*

2.2 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.

Casts We start by defining upcasts and downcasts, using type and term dynamism in Figure 5. Given that $A_0 \sqsubseteq A_1$, the upcast is a function from A_0 to A_1 such that for any $t : A_0$, $\langle A_1 \leftarrow\!\!\prec A_0 \rangle t$ is the *least dynamic term of type* A_1 *that is more dynamic than* t. The UR rule can be thought of as the "introduction rule", saying $\langle A' \leftarrow\!\!\prec A \rangle x$ 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 $\langle A' \leftarrow\!\!\prec A \rangle x$ — since $\langle A' \leftarrow\!\!\prec A \rangle x$ is the *least* dynamic term with this property. The rules for projections are dual, ensuring that for $x' : A'$, $\langle A \twoheadleftarrow A' \rangle x'$ is the most dynamic term of type A that is less dynamic than x'.

In fact, combined with the TmPrec-Trans rule, we can show that it has a slightly more general property: $\langle A' \leftarrow\!\!\prec A \rangle x$ 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 TmPrec-Trans and TmPrec-Comp, assuming $A \sqsubseteq A'$ and $A' \sqsubseteq A''$:

$$\frac{\Phi \vdash t \sqsubseteq t' : A \sqsubseteq A'}{\Phi \vdash t \sqsubseteq \langle A'' \leftarrow\!\!\prec A' \rangle t' : A \sqsubseteq A''} \text{ Up-R} \qquad \frac{\Phi \vdash t \sqsubseteq t'' : A \sqsubseteq A''}{\Phi \vdash \langle A' \leftarrow\!\!\prec A \rangle t \sqsubseteq t'' : A' \sqsubseteq A''} \text{ Up-L}$$

$$\frac{\Phi \vdash t' \sqsubseteq t'' : A' \sqsubseteq A''}{\Phi \vdash \langle A \twoheadleftarrow A' \rangle t' \sqsubseteq t'' : A \sqsubseteq A''} \text{ Dn-L} \qquad \frac{\Phi \vdash t \sqsubseteq t'' : A \sqsubseteq A''}{\Phi \vdash t \sqsubseteq \langle A' \twoheadleftarrow A'' \rangle t'' : A \sqsubseteq A'} \text{ Dn-R}$$

In particular, the upcast is left-invertible, and the downcast is right-invertible (which agrees with their status and left and right adjoints discussed below). However, when reasoning "bottom-up", the presuppositions the conclusion of Up-R ($A \sqsubseteq A''$ and $A' \sqsubseteq A''$) do not entail $A \sqsubseteq A'$, and the presuppositions of the conclusion of Dn-L ($A \sqsubseteq A'$ and $A \sqsubseteq A''$) do not entail $A' \sqsubseteq A''$. Thus, the premise suffices *whenever it is well-formed*, but it might not be.

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 $\langle A' \leftarrow\!\!\prec A \rangle \langle A \twoheadleftarrow A' \rangle t \sqsubseteq t$ and $t \sqsubseteq \langle A \twoheadleftarrow A' \rangle \langle A' \leftarrow\!\!\prec A \rangle t$. However in programming practice, 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 $t \sqsupseteq\sqsubseteq \langle A \twoheadleftarrow A' \rangle \langle A' \leftarrow\!\!\prec A \rangle t$. We can restrict to Galois insertions by adding the *retract axiom* RetractAx. 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.

$$\frac{\Gamma \vdash t : A \qquad A \sqsubseteq A'}{\Gamma \vdash \langle A' \leftarrowtail A \rangle t : A'} \text{ Upcast} \qquad \frac{\Gamma \vdash t : A' \qquad A \sqsubseteq A'}{\Gamma \vdash \langle A \twoheadleftarrow A' \rangle t : A} \text{ Downcast}$$

$$\frac{A \sqsubseteq A'}{x \sqsubseteq x : A \sqsubseteq A \vdash x \sqsubseteq \langle A' \leftarrowtail A \rangle x : A \sqsubseteq A'} \text{ UR} \qquad \frac{A \sqsubseteq A'}{x' \sqsubseteq x' : A' \sqsubseteq A' \vdash \langle A \twoheadleftarrow A' \rangle x' \sqsubseteq x' : A \sqsubseteq A'} \text{ DL}$$

$$\frac{A \sqsubseteq A'}{x \sqsubseteq x' : A \sqsubseteq A' \vdash \langle A' \leftarrowtail A \rangle x \sqsubseteq x' : A' \sqsubseteq A'} \text{ UL} \qquad \frac{A \sqsubseteq A'}{x \sqsubseteq x' : A \sqsubseteq A' \vdash x \sqsubseteq \langle A \twoheadleftarrow A' \rangle x' : A \sqsubseteq A} \text{ DR}$$

$$\frac{A \sqsubseteq A'}{x : A \sqsubseteq x : A \vdash \langle A \twoheadleftarrow A' \rangle \langle A' \leftarrowtail A \rangle x \sqsubseteq x : A} \text{ RetractAx} \qquad \frac{}{A \sqsubseteq ?} \qquad \frac{}{\Gamma \vdash \mho_A : A} \qquad \frac{\Phi : \Gamma \sqsubseteq \Gamma}{\Phi \vdash \mho_A \sqsubseteq t : A}$$

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

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. The type error, written as $\mho$, is defined by the fact that it is a constant at every type that is a least element of that type. By transitivity, this further implies that $\mho_A \sqsubseteq t : A \sqsubseteq A'$ for any $A' \sqsupseteq A$.

Negative Connectives Next we illustrate how simple *negative* types can be defined in preorder type theory. Specifically, we present the unit type, products and function types in 6. The type and term constructors are the same as those in the simply typed λ-calculus. For type dynamism, we make every connective *monotone* in every argument, including the function type. Due to the covariance of the function type, type dynamism is sometimes naïvely referred to as "naïve subtyping"; see 5 for a semantic intuition. 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 write $\sqsupseteq\sqsubseteq$ to mean a rule exists in each direction (which requires that the types and contexts are also equi-dynamic).

Gradual Type Theory and Signatures 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 (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 the many consequences of the simple axioms of gradual type theory. We show that almost every reduction in an operational presentation of call-by-name gradual typing, and many principles used in optimization of implementations, are justified by the universal property for casts in all types, the β, η rules, and the congruence rules for connectives and terms. Thus, the combination of graduality and η principles is a strong specification for gradual typing and considerably narrows the design space. We summarize these derivations in the following theorem:

Theorem 1. *In Gradual Type Theory, all of the following are derivable whenever the upcasts, downcasts are well-formed.*

1. *Universal Property: Casts are unique up to* $\sqsupseteq\sqsubseteq$*.*
2. *Identity:* $\langle A \leftarrowtail A \rangle t \sqsupseteq\sqsubseteq t$ *and* $\langle A \twoheadleftarrow A \rangle t \sqsupseteq\sqsubseteq t$*.*

$$\frac{A \sqsubseteq A' \qquad B \sqsubseteq B'}{A \to B \sqsubseteq A' \to B'} \qquad \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x : A.t : A \to B} \qquad \frac{\Gamma \vdash t : A \to B \qquad \Gamma \vdash u : A}{\Gamma \vdash t\, u : B}$$

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

$$\frac{}{\Gamma \vdash (\lambda x : A.t)u \sqsupseteq\sqsubseteq t[u/x] : B} \qquad \frac{}{\Gamma \vdash t \sqsupseteq\sqsubseteq (\lambda x : A.t\, x) : A \to B \sqsubseteq A \to B}$$

$$\frac{A_1 \sqsubseteq A_1' \qquad A_2 \sqsubseteq A_2'}{A_1 \times A_2 \sqsubseteq A_1' \times A_2'} \qquad \frac{\Gamma \vdash t_1 : A_1 \qquad \Gamma \vdash t_2 : A_2}{\Gamma \vdash (t_1, t_2) : A_1 \times A_2} \qquad \frac{\Gamma \vdash t : A_1 \times A_2 \qquad i \in 1, 2}{\Gamma \vdash \pi_i t : A_i}$$

$$\frac{\Phi \vdash t_1 \sqsubseteq t_1' : A_1 \sqsubseteq A_1' \qquad \Phi \vdash t_2 \sqsubseteq t_2' : A_2 \sqsubseteq A_2'}{\Phi \vdash (t_1, t_2) \sqsubseteq (t_1', t_2') : A_1 \times A_2 \sqsubseteq A_1' \times A_2'} \qquad \frac{\Phi \vdash t \sqsubseteq t' : A_1 \times A_2 \sqsubseteq A_1' \times A_2'}{\Phi \vdash \pi_i t \sqsubseteq \pi_i t' : A_i \sqsubseteq A_i'}$$

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

$$\frac{}{\Gamma \vdash () : 1} \qquad \frac{}{\Gamma \vdash t \sqsupseteq\sqsubseteq () : 1}$$

Figure 6: Simple Negative Types

3. *Composition:* $\langle A'' \leftarrow A \rangle t \sqsupseteq\sqsubseteq \langle A'' \leftarrow A' \rangle \langle A' \leftarrow A \rangle t$ *and* $\langle A \twoheadleftarrow A'' \rangle t \sqsupseteq\sqsubseteq \langle A \twoheadleftarrow A' \rangle \langle A' \twoheadleftarrow A'' \rangle t$.
4. *Function Cast Reduction:* $\langle A' \to B' \leftarrow A \to B \rangle t \sqsupseteq\sqsubseteq \lambda x : A'.\langle B' \leftarrow B \rangle (t(\langle A \twoheadleftarrow A' \rangle x))$ *and* $\langle A \to B \twoheadleftarrow A' \to B' \rangle t \sqsupseteq\sqsubseteq \lambda x : A'.\langle B \twoheadleftarrow B' \rangle (t(\langle A' \leftarrow A \rangle x))$.
5. *Product Cast Reduction:* $\langle A_0' \times A_1' \leftarrow A_0 \times A_1 \rangle t \sqsupseteq\sqsubseteq (\langle A_0' \leftarrow A_0 \rangle \pi_0 t, \langle A_1' \leftarrow A_1 \rangle \pi_1 t)$ *and* $\langle A_0 \times A_1 \twoheadleftarrow A_0' \times A_1' \rangle t \sqsupseteq\sqsubseteq (\langle A_0 \twoheadleftarrow A_0' \rangle \pi_0 t, \langle A_1 \twoheadleftarrow A_1' \rangle \pi_1 t)$.
6. *Adjunction:* $t \sqsubseteq \langle A \twoheadleftarrow A' \rangle \langle A' \leftarrow A \rangle t$ *and* $\langle A' \leftarrow A \rangle \langle A \twoheadleftarrow A' \rangle t \sqsubseteq t$*, for which the retract axiom is the converse.*
7. *Cast Congruence:* $x \sqsubseteq y : A \sqsubseteq B \vdash \langle A' \leftarrow A \rangle x \sqsubseteq \langle B' \leftarrow B \rangle y : A' \sqsubseteq B'$ *and* $x' \sqsubseteq y' : A' \sqsubseteq B' \vdash \langle A \twoheadleftarrow A' \rangle x' \sqsubseteq \langle B \twoheadleftarrow B' \rangle y' : A \sqsubseteq B$.
8. *Errors:* $\langle A' \leftarrow A \rangle \mho_A \sqsupseteq\sqsubseteq \mho_{A'}$*, and by the retract axiom* $\langle A \twoheadleftarrow A' \rangle \mho_A \sqsupseteq\sqsubseteq .\mho_{A'}$.
9. *Equi-dynamism implies isomorphism: If* $A \sqsupseteq\sqsubseteq B$*, then* A *is isomorphic to* B.

Many of these facts are usually given as part of the *definition* of the operational semantics of the language, but we show here that they are uniquely determined by the other principles of gradual type theory, the specification for upcasts/downcasts, the congruence rules for term dynamism and the η equivalence principles for types. This shows that the combination of graduality and η principles is a strong specification for gradual typing and considerably narrows the design space. We can also weaken results to an ordering if η only holds in one direction.

Uniqueness of Casts First, to prove that casts are unique, suppose that there was a second version of the upcast $\langle A' \leftarrow A \rangle t$ with analogous Up-L' and Up-R'. Then we can show that this 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.

$$\dfrac{\dfrac{x : A \vdash x \sqsubseteq x : A}{x : A \vdash x \sqsubseteq \langle A' \hookleftarrow A\rangle x : A \sqsubseteq A'}\ \textsc{Up-R'}}{x : A \vdash \langle A' \leftharpoonup A\rangle x \sqsubseteq \langle A' \hookleftarrow A\rangle x : A'}\ \textsc{Up-R} \qquad \dfrac{\dfrac{x : A \vdash x \sqsubseteq x : A}{x : A \vdash x \sqsubseteq \langle A' \leftharpoonup A\rangle x : A \sqsubseteq A'}\ \textsc{Up-R}}{x : A \vdash \langle A' \hookleftarrow A\rangle x \sqsubseteq \langle A' \leftharpoonup A\rangle x : A'}\ \textsc{Up-R'}$$

By duality, the same holds for the downcast.

Identity Casts The upcast and downcast from a type to itself are the identity function. 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 $\langle A \hookleftarrow A\rangle x$ are equi-precise and each direction is an instance of Up-L or Up-R.

$$\dfrac{}{x : A \vdash x \sqsubseteq \langle A \hookleftarrow A\rangle x : A}\ \textsc{UR} \qquad \dfrac{}{x : A \sqsubseteq x : A \vdash \langle A \hookleftarrow A\rangle x \sqsubseteq x : A \sqsubseteq A}\ \textsc{UL}$$

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

$$\dfrac{}{x : A \sqsubseteq x' : A \vdash \langle B \hookleftarrow A\rangle x \sqsubseteq x' : B \sqsubseteq A}\ \textsc{not an instance of UL}$$

because the conclusion violates the presupposition of the judgment, which would require $B \sqsubseteq A$, and is moreover not an instance of Up-L, which would require x' to have type B, 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.

The downcast case has a perfectly dual proof.

Composition of Casts 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:

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

More generally, it implies that casts from A to B commute over the dynamic type, e.g. $\langle ? \hookleftarrow B\rangle\langle B \hookleftarrow A\rangle x \sqsupseteq\sqsubseteq \langle ? \hookleftarrow A\rangle x$—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.

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

$$\dfrac{\dfrac{\dfrac{}{x : A \vdash x \sqsubseteq \langle B \hookleftarrow A\rangle x : A \sqsubseteq B}\ \textsc{Up-R}}{x : A \vdash x \sqsubseteq \langle C \hookleftarrow B\rangle\langle B \hookleftarrow A\rangle x : A \sqsubseteq C}\ \textsc{Up-R}}{x : A \vdash \langle C \hookleftarrow A\rangle x \sqsubseteq \langle C \hookleftarrow B\rangle\langle B \hookleftarrow A\rangle x : C}\ \textsc{Up-L}$$

$$\dfrac{\dfrac{\dfrac{}{x : A \vdash x \sqsubseteq \langle C \hookleftarrow A\rangle x : A \sqsubseteq C}\ \textsc{Up-R}}{x : A \vdash \langle B \hookleftarrow A\rangle x \sqsubseteq \langle C \hookleftarrow A\rangle x : B \sqsubseteq C}\ \textsc{Up-L}}{x : A \vdash \langle C \hookleftarrow B\rangle\langle B \hookleftarrow A\rangle x \sqsubseteq \langle C \hookleftarrow A\rangle x : C}\ \textsc{Up-L}$$

Casts are Galois Connections Next, as mentioned previously, we show that the specifications of the upcast and downcast make them into a *Galois connection*/adjunction with the upcast as the upper/left adjoint. This tells us that given $A \sqsubseteq A'$, the "round-trip" from A' down to A and back results in a less dynamic term and the other round-trip results in a more dynamic term. In programming practice, we expect the 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*.

$$\dfrac{\dfrac{x : A \vdash x \sqsubseteq x : A \sqsubseteq A}{x : A \vdash x \sqsubseteq \langle A' \leftharpoondown A \rangle x : A \sqsubseteq A'}\ \textsc{Up-L}}{x : A \vdash x \sqsubseteq \langle A \twoheadleftarrow A' \rangle \langle A' \leftharpoondown A \rangle x : A}\ \textsc{Dn-R} \qquad \dfrac{\dfrac{x' : A' \vdash x' \sqsubseteq x' : A' \sqsubseteq A'}{x' : A' \vdash \langle A \twoheadleftarrow A' \rangle x' \sqsubseteq x' : A \sqsubseteq A'}\ \textsc{Dn-L}}{x' : A' \vdash \langle A' \leftharpoondown A \rangle \langle A \twoheadleftarrow A' \rangle x' \sqsubseteq x' : A'}\ \textsc{Up-L}$$

Cast Congruence and the Gradual Guarantee Recall that the gradual guarantee [28] 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: When $A \sqsubseteq A', B \sqsubseteq B', A \sqsubseteq B, A' \sqsubseteq B'$,

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

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

This is a congruence rule for casts in the type positions on type dynamism relations $A \sqsubseteq B$ and $A' \sqsubseteq B'$. The proof of the first is

$$\dfrac{\dfrac{x \sqsubseteq y : A \sqsubseteq B \vdash x \sqsubseteq y : A \sqsubseteq B}{x \sqsubseteq y : A \sqsubseteq B \vdash x \sqsubseteq \langle B' \leftharpoondown B \rangle y : A \sqsubseteq B'}\ \textsc{Up-R}}{x \sqsubseteq y : A \sqsubseteq B \vdash \langle A' \leftharpoondown A \rangle x \sqsubseteq \langle B' \leftharpoondown B \rangle y : A' \sqsubseteq B'}\ \textsc{Up-L}$$

and the second is dual.

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

Strictness of Casts Next we show that upcasts and downcasts are *strict* with respect to the type error $\mho$. The upcast preserves $\mho$ because it is a left/upper adjoint and therefore preserves colimits/joins like $\mho$. More concretely, $\langle A' \leftharpoondown A \rangle \mho_A \sqsubseteq \mho_{A'}$ because $\mho_{A'}$ is more dynamic than $\mho_A$ and is the least dynamic term of type A' so is in particular less dynamic than anything more dynamic than $\mho_A$. As derivations:

$$\dfrac{}{\cdot \vdash \mho_{A'} \sqsubseteq \langle A' \leftharpoondown A \rangle \mho_A : A'}\ \textsc{Err-Bot} \qquad \dfrac{\dfrac{}{\cdot \vdash \mho_A \sqsubseteq \mho_{A'} : A \sqsubseteq A'}\ \textsc{Err-Bot'}}{\cdot \vdash \langle A' \leftharpoondown A \rangle \mho_A \sqsubseteq \mho_{A'} : A'}\ \textsc{Up-L}$$

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

Function and Product Casts In this section we derive the standard "wrapping" implementations from [11, 10] for the function and product casts, i.e., casts derived from dynamism derivations of $A \to B \sqsubseteq A' \to B'$ and $A_0 \times A_1 \sqsubseteq A_0' \times A_1'$. A function upcast uses the downcast on inputs and upcast on outputs and vice-versa for the downcast. This shows that the standard implementation is in fact the *unique* implementation to satisfy soundness and graduality.

Each proof is modular (depends on no more type or term constructors other than those related to function and product types respectively). To make the proofs shorter, we first derive a higher-level "extensionality

principle" for each: a function is less dynamic than another if applying it to a less dynamic input yields a less dynamic result and a product is less dynamic than another if its components are:

$$\frac{\Phi, x \sqsubseteq x' : A \sqsubseteq A' \vdash tx \sqsubseteq t'x' : B \sqsubseteq B'}{\Phi \vdash t \sqsubseteq t' : A \to B \sqsubseteq A' \to B'}\ \textsc{Fun-Ext} \qquad \frac{\forall i \in \{0,1\}.\Phi \vdash \pi_i t \sqsubseteq \pi_i t' : A_i \sqsubseteq A_i'}{\Phi \vdash t \sqsubseteq t' : A_0 \times A_1 \sqsubseteq A_0' \to A_1'}\ \textsc{Prod-Ext}$$

Both follow from the η principles for each type and the congruence rules for the introduction forms:

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

$$\frac{t \sqsubseteq (\pi_0 t, \pi_1 t) \qquad (\pi_0 t', \pi_1 t') \sqsubseteq t' \qquad \dfrac{\forall i \in \{0,1\}.\Phi \vdash \pi_i t \sqsubseteq \pi_i t' : A_i \sqsubseteq A_i'}{\Phi \vdash (\pi_0 t, \pi_1 t) \sqsubseteq (\pi_0 t', \pi_1 t') : A_0 \times A_1 \sqsubseteq A_0' \to A_1'}}{\Phi \vdash t \sqsubseteq t' : A_0 \times A_1 \sqsubseteq A_0' \to A_1'}\ \textsc{TmPrec-Trans}$$

For the function contract, the standard implementation is

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

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 $x \sqsubseteq x' : A \sqsubseteq A' \vdash fx \sqsubseteq \langle B' \leftarrow B \rangle (f(\langle A \twoheadleftarrow A' \rangle x'))$. This follows from congruence of application and the rules of casts. The other direction is essentially the dual.

In fact, a more direct, but less high-level proof is possible that shows that each direction of the equivalence $\langle A' \to B' \leftarrow A \to B \rangle f \sqsubseteq \lambda x' : A'.\langle B' \leftarrow B \rangle (f(\langle A \twoheadleftarrow A' \rangle x'))$ only relies on the corresponding direction of the η principle $f \sqsupseteq\sqsubseteq \lambda x.fx$, and we don't need to use β equivalence at all. In Figure 7, we show the direct proofs of both directions of $\sqsupseteq\sqsubseteq$ for the upcasts of function and product types; the downcast proofs are dual.

Disjointness of Types and Equi-Dynamism versus Isomorphism Finally, 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[1]. First, *equi-dynamism* $A \sqsupseteq\sqsubseteq B$ means $A \sqsubseteq B$ and $B \sqsubseteq A$. Second, isomorphism means functions $f : A \to B$ and $g : B \to A$ such that $f \circ g \sqsupseteq\sqsubseteq (\lambda x : B.x)$ and $g \circ f \sqsupseteq\sqsubseteq (\lambda x : A.x)$. If two types are equi-dynamic, then any casts between them are an isomorphism: If $A \sqsupseteq\sqsubseteq B$, then

- $x.\langle B \leftarrow A \rangle x$ and $y.\langle A \leftarrow B \rangle y$ form an isomorphism of types.

$$\frac{\dfrac{x : A \vdash x \sqsubseteq x : A}{x : A \vdash \langle B \leftarrow A \rangle x \sqsubseteq x : B \sqsubseteq A}}{x : A \vdash \langle A \leftarrow B \rangle \langle B \leftarrow A \rangle x \sqsubseteq x : A} \qquad \frac{x : A \vdash x \sqsubseteq \langle B \leftarrow A \rangle x : A \sqsubseteq B}{x : A \vdash x \sqsubseteq \langle A \leftarrow B \rangle \langle B \leftarrow A \rangle x : A}$$

- $x.\langle A \twoheadleftarrow B \rangle x$ and $y.\langle B \twoheadleftarrow A \rangle y$ form an isomorphism of types. This is dual to the previous part.
- $x.\langle B \leftarrow A \rangle x$ and $y.\langle A \twoheadleftarrow B \rangle y$ form an isomorphism of types.

$$\frac{\dfrac{x : A \vdash x \sqsubseteq x : A}{x : A \vdash \langle B \leftarrow A \rangle x \sqsubseteq x : B \sqsubseteq A}}{x : A \vdash \langle A \twoheadleftarrow B \rangle \langle B \leftarrow A \rangle x \sqsubseteq x : A} \qquad \frac{x : A \vdash x \sqsubseteq \langle B \leftarrow A \rangle x : A \sqsubseteq B}{x : A \vdash x \sqsubseteq \langle A \twoheadleftarrow B \rangle \langle B \leftarrow A \rangle x : A}$$

- $x \vdash \langle A \leftarrow B \rangle x \sqsupseteq\sqsubseteq \langle A \twoheadleftarrow B \rangle x$. This follows from uniqueness of inverses (which is true by the usual argument) and the previous two.

[1] Corresponding to the two notions of isomorphism in double categories

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

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

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

$$\dfrac{\dfrac{p \sqsubseteq (\pi_0 p, \pi_1 p) \qquad \dfrac{\dfrac{\dfrac{\pi_i p \sqsubseteq \pi_i p}{\forall i \in 0, 1.\, \pi_i p \sqsubseteq \langle A'_i \hookleftarrow A_i \rangle \pi_i p}}{p \vdash (\pi_0 p, \pi_1 p) \sqsubseteq (\langle A'_0 \hookleftarrow A_0 \rangle \pi_0 p, \langle A'_1 \hookleftarrow A_1 \rangle \pi_0 p)}}{p \vdash p \sqsubseteq (\langle A'_0 \hookleftarrow A_0 \rangle \pi_0 p, \langle A'_1 \hookleftarrow A_1 \rangle \pi_0 p)}}{p : A_0 \times A_1 \vdash \langle A'_0 \times A'_1 \hookleftarrow A_0 \times A_1 \rangle p \sqsubseteq (\langle A'_0 \hookleftarrow A_0 \rangle \pi_0 p, \langle A'_1 \hookleftarrow A_1 \rangle \pi_0 p) : A'_0 \times A'_1}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{p \sqsubseteq p}{p \sqsubseteq \langle A'_0 \times A'_1 \hookleftarrow A_0 \times A_1 \rangle p}}{\pi_i p \sqsubseteq \pi_i \langle A'_0 \times A'_1 \hookleftarrow A_0 \times A_1 \rangle p}}{\forall i \in 0, 1.\, \langle A'_i \hookleftarrow A_i \rangle \pi_i p \sqsubseteq \pi_i \langle A'_0 \times A'_1 \hookleftarrow A_0 \times A_1 \rangle p}}{(\langle A'_0 \hookleftarrow A_0 \rangle \pi_0 p, \langle A'_1 \hookleftarrow A_1 \rangle \pi_0 p) \sqsubseteq (\pi_0 \langle A'_0 \times A'_1 \hookleftarrow A_0 \times A_1 \rangle p, \pi_1 \langle A'_0 \times A'_1 \hookleftarrow A_0 \times A_1 \rangle p)}\ \mathcal{E}$$

$$\dfrac{\mathcal{E} \qquad (\pi_0 t, \pi_1 t) \sqsubseteq t \text{ with } t = \langle A'_0 \times A'_1 \hookleftarrow A_0 \times A_1 \rangle p}{p : A_0 \times A_1 \vdash (\langle A'_0 \hookleftarrow A_0 \rangle \pi_0 p, \langle A'_1 \hookleftarrow A_1 \rangle \pi_0 p) \sqsubseteq \langle A'_0 \times A'_1 \hookleftarrow A_0 \times A_1 \rangle p : A'_0 \times A'_1}$$

Figure 7: Negative Type Upcast Implementations

The converse, 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 $\langle (C \times D) \nleftarrow ? \rangle \langle ? \leftarrow (A \to B) \rangle x \sqsubseteq \mho$ 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 $\langle B \nleftarrow ? \rangle \langle ? \leftarrow A \rangle x$ should succeed, not fail. If it fails, then every term of A, B equals $\mho$: Assume $A \sqsupseteq\sqsubseteq B$ and $\langle B \nleftarrow ? \rangle \langle ? \leftarrow A \rangle x \sqsubseteq \mho$. By composition and the adjunction property

$$\langle B \leftarrow A \rangle x \sqsubseteq \langle B \nleftarrow ? \rangle \langle ? \leftarrow B \rangle \langle B \leftarrow A \rangle x \sqsupseteq\sqsubseteq \langle B \nleftarrow ? \rangle \langle ? \leftarrow A \rangle x \sqsubseteq \mho$$

But by above, $\langle A \leftarrow B \rangle$ is an isomorphism, so

$$x : A \sqsupseteq\sqsubseteq \langle A \nleftarrow B \rangle \langle B \leftarrow A \rangle x \sqsubseteq \langle B \nleftarrow A \rangle \mho \sqsupseteq\sqsubseteq \mho$$

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 soundness and completeness (i.e. initiality) theorems. 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 in *preorder categories*, which are categories internal to the category of preorders.[2] 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 also 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 3 (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"* $i : \mathbb{C}_0 \to \mathbb{C}_1$
4. *A monotone composition function* $\circ : \mathbb{C}_1 \times_{\mathbb{C}_0} \mathbb{C}_1 \to \mathbb{C}_1$*, i.e., a monotone function that takes for any* $f, g \in \mathbb{C}_1$ *with* $sf = tg$*, a morphism* $f \circ g$ *with* $s(f \circ g) = sg$ *and* $t(f \circ g) = tf$*.*
5. *unitality and associativity laws for composition:* $f \circ i(A) = f$*,* $i(B) \circ f = f$ *and* $(f \circ g) \circ h = f \circ (g \circ h)$*.*

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 classically resolved by assuming that models have product types and using categorical products to interpret the context [17]. However, we take a more modern *multicategorical* view, in which our notion of model will axiomatize algebraically a notion of morphism with many inputs. Using terminology from [5], we define a model of preorder type theory as a "virtually" cartesian preorder category, which does not necessarily have product *objects*, but whose morphisms' source is a "virtual" product of objects, i.e. a context.

[2] 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 objects are not.

Definition 4 (Virtually Cartesian Preorder Category)**.** *A virtually cartesian preorder category (VCP category)* $\mathbb{C}$ *consists of*

1. *a preordered set of "objects"* $\mathbb{C}_0$
2. *a preordered set of "multiarrows"* $\mathbb{C}_1$
3. *Monotone functions of "source"* $s : \mathbb{C}_1 \to Ctx(\mathbb{C})_0$*, "target"* $\mathbb{C}_1 \to \mathbb{C}_0$*.*
4. *A monotone function of "identity"/"projection"* $x : Ctx(\mathbb{C})_0 \times \mathbb{C}_0 \times Ctx(\mathbb{C})_0 \to \mathbb{C}_1$ *satisfying* $s(x(\Gamma, A, \Delta)) = \Gamma, A, \Delta$ *and* $t(x(\Gamma, A, \Delta)) = A$
5. *A monotone "composition" function* $\circ : \mathbb{C}_1 \times_{Ctx(\mathbb{C})_0} Ctx(\mathbb{C})_1 \to \mathbb{C}_1$ *satisfying* $s(f \circ \gamma) = s(\gamma)$ *and* $t(f \circ \gamma) = t(f)$
6. *Satisfying "Identity"/ "Projection" laws:*
$$f \circ id = f \quad x(\Gamma, A, \Delta) \circ \gamma = \gamma(|\Gamma|)$$
7. *Satisfying the "Associativity" law:*
$$(f \circ \gamma) \circ \delta = f \circ (\gamma \circ \delta)$$
8. *for every* $\Gamma, \Delta \in Ctx(\mathbb{C})_0$ *and* $A \in \mathbb{C}_0$ *and morphism* $f \in \mathbb{C}_1(\Gamma; A)$ *and substitution* $\gamma \in Ctx(\mathbb{C})_1(\Delta; \Gamma)$*, a composite* $f \circ \gamma$*.*

where we simultaneously define $Ctx(\mathbb{C})$ *by*

1. $Ctx(\mathbb{C})_0$ *is the set of lists of elements of* $\mathbb{C}_0$ *(called "contexts") with the point-wise preorder.*
2. *A substitution* $Ctx(\mathbb{C})_1(\Gamma; A_1, \ldots, A_n)$ *is a function* γ *that assigns for every* $i \in \{1, \ldots, n\}$ *a multiarrow* $\gamma(i) \in \mathbb{C}_1(\Gamma; A_i)$*, with pointwise ordering. Then* $Ctx(\mathbb{C})_1$ *is the set of triples of two contexts and a substitution between them, with pointwise ordering.*
3. *Substitutions are composed as follows. Given* $\gamma \in Ctx(\mathbb{C})(\Delta, \Gamma)$ *and* $\gamma' \in Ctx(\mathbb{C})(\Gamma, A_1, \ldots, A_n)$*, define* $(\gamma' \circ \gamma)(i) = \gamma'(i) \circ \gamma$ *where the latter* $\circ$ *is composition of a substitution with a multiarrow.*
4. *The identity substitution* $Ctx(\mathbb{C})$ *is given by the pointwise identity morphism.*

Note that the axioms of multiarrow composition with substitutions are precisely what is needed to make the definition of identity and composition for Ctx($\mathbb{C}$) into a preorder category.

Next, we present the soundness and completeness theorems of the interpretation of preorder type theory in a preorder category. Soundness informally means that any interpretation of the base types, function symbols and axioms of PTT can be extended to a compositional semantics in which all derivable theorems are true.

Definition 5 (Interpretation of Preorder Type Theory/Soundness)**.** *We define a sequence of denotation functions* $[\![\cdot]\!]$ *interpreting the syntax of preorder type theory in a preorder category* C*. Let* $\Sigma = (\Sigma_0, \Sigma_1, \Sigma_2, \Sigma_3)$ *be a PTT signature, and all syntax be relative to* Σ*.*

1. *Given a base-type interpretation* $(\!|\cdot|\!) : \Sigma_0 \to \mathbb{C}_0$*, we extend it to a type interpretation function* $[\![\cdot]\!] : Type \to \mathbb{C}_0$*. This is trivial since PTT only has base types.*
2. *If for every* $(A, B) \in \Sigma_1$*,* $[\![A]\!] \sqsubseteq [\![B]\!]$ *holds, then for any derivable* $A \sqsubseteq B$*,* $[\![A]\!] \sqsubseteq [\![B]\!]$ *holds. The reflexivity and transitivity rules hold in a preorder.*
3. *Given an interpretation of the function symbols* $(\!|\cdot|\!) : \Sigma_2(A_1, \ldots, A_n; B) \to \mathbb{C}_1([\![A_1]\!], \ldots, [\![A_n]\!]; [\![B]\!])$ *we can extend it to a compositional a semantics function* $[\![\cdot]\!] : x_1 : A_1 \ldots, x_n : A_n \vdash \cdot : B \to \mathbb{C}_1([\![A_1]\!], \ldots, [\![A_n]\!]; [\![B]\!])$ *in that:*
$$[\![f(x_1, \ldots, x_n)]\!] = (\!|f|\!)$$
$$[\![t[\gamma]]\!] = [\![t]\!] \circ [\![\gamma]\!]$$

where we simultaneously define semantics of substitutions as $[\![\gamma]\!] = [\![\cdot]\!] \circ \gamma$. *This is defined as:*

$$[\![\Gamma, x : A, \Delta \vdash x : A]\!] = x(\lfloor \Gamma \rfloor, A, \lfloor \Delta \rfloor)$$
$$[\![\Gamma \vdash f(\gamma(x_1), \ldots, \gamma(x_n))]\!] = (\!|f|\!) \circ [\![\gamma]\!]$$

where $\lfloor x_1 : A_1, \ldots, x_n : A_n \rfloor = A_1, \ldots, A_n$.

4. *If for every* $(t, t') \in \Sigma_3$, $[\![t]\!] \sqsubseteq [\![t']\!]$ *holds, then for any derivation* $\Phi \vdash t \sqsubseteq t' : A \sqsubseteq A'$, $[\![t]\!] \sqsubseteq [\![t']\!]$ *holds. The four rules* (Var, Comp, Refl, Trans) *for term dynamism are a syntactic presentation of the horizontal and vertical identity and composition operations for squares in a double category, and thus hold in any preorder category.*

Next, completeness informally means that if a theorem is true in every model, then it is derivable in the syntax. We prove it in the standard method for categorical models, which is to show that the *syntax* presents a preorder category where the true theorems are exactly the derivable theorems.

Theorem 2 (Completeness of Preorder Category Semantics)**.** *Let* Σ *be a PTT signature and let all syntax be relative to that signature.*

1. *For any two types* A, B, *if for every interpretation of* Σ $(\!|\cdot|\!) \to C$, $[\![A]\!] \sqsubseteq [\![B]\!]$ *holds then it is derivable that* $A \sqsubseteq B$.
2. *For any two terms* t, t' *relative to* Σ, *if for every interpretation* $[\![t]\!] \sqsubseteq [\![t']\!]$ *then it is derivable in PTT that* $t \sqsubseteq t'$.

Proof. We construct the preorder category $\mathrm{PTT}(\Sigma)$ as follows:

1. The objects are the types generated by Σ.
2. $A \sqsubseteq B$ holds when $A \sqsubseteq B$ is derivable.
3. A term $\mathrm{PTT}(\Sigma)(A_1, \ldots, A_n; B)$ is a term $x_1 : A_1, \ldots, x_n : A_n \vdash t : B$ for some variables $x_1, \ldots, x_n$, 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 VCP category involves the standard proofs of the associativity and unitality of substitution and an easy proof that substitution is monotone with respect to term dynamism. □

4.1 Gradual Typing Structures

Next, we describe the additional structure on a VCP category to model full gradual type theory: casts are modeled by an *equipment* [27], a dynamic type by a greatest object, and the type error by a least element of every hom-set.

Definition 6 (Equipment [27])**.** *A VCP category* $\mathbb{C}$ *is an* equipment *if for every* $A \sqsubseteq B$, *there exist morphisms* $u_{A,B} \in \mathbb{C}(A, B)$ *and* $d_{A,B} \in \mathbb{C}(B, A)$ *such that the following hold:*

1. $u_{A,B} \sqsubseteq id_B$
2. $id_A \sqsubseteq u_{A,B}$
3. $d_{A,B} \sqsubseteq id_B$
4. $id_A \sqsubseteq d_{A,B}$

An equipment is coreflective *if also* $d_{A,B} \circ u_{A,B} \sqsubseteq id_A$.

Definition 7 (Greatest Object)**.** *A greatest object in a VCP category* $\mathbb{C}$ *is a greatest element of the preorder of objects* $\mathbb{C}_0$.

Definition 8 (Local Bottoms)**.** *A VCP category* $\mathbb{C}$ *has local bottoms if every hom set* $\mathbb{C}(A_1, \ldots, A_n; B)$ *has a least element* $\bot$ *and for every substitution* $\gamma \in Ctx(\mathbb{C})_1(B_1 \ldots, B_m; A_1, \ldots, A_n)$ *we have* $\bot \circ \gamma \sqsupseteq \bot$.

Interpreting Negative Types Next, we define a *cartesian closed VCP category*, which will model negative function and product types. While we use the adjectives "closed" and "cartesian", the structure exhibited here is only unique up to canonical *isomorphism*, and the objects are *not* unique up to *order-equivalence* (equi-dynamism). Thus, there may be *different* order-inequivalent ways that a VCP category can be closed or cartesian, so it is important that e.g. a closed VCP category is a VCP category *with* a *choice* of exponentials.

Definition 9 (Closed VCP Category)**.** *A Closed VCP category is a VCP category* $\mathbb{C}$ *with a monotone function on objects* $\to\colon \mathbb{C}_0^2 \to \mathbb{C}_0$ *making for every pair of objects* $X, Y \in \mathbb{C}$ *an "exponential" object* $X \to Y$ *with a monotone function*

$$\lambda : \mathbb{C}(\Gamma, X; Y) \to \mathbb{C}(\Gamma; X \to Y)$$

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

$$\lambda(h) \circ \gamma \sqsupseteq\sqsubseteq \lambda(h \circ (\gamma, x(\Gamma, X, \cdot)))$$

with a morphism

$$app \in \mathbb{C}(X \to Y, X; Y)$$

such that the function given by

$$f \mapsto app \circ (f, x(X)) : \mathbb{C}(\Gamma; X \to Y) \to \mathbb{C}(\Gamma, X; Y)$$

is an inverse to λ *up to* $\sqsupseteq\sqsubseteq$*.*

Definition 10 (Cartesian VCP Category)**.** *A Cartesian VCP category is a VCP category* $\mathbb{C}$ *with a monotone function* $\times : \mathbb{C}_0^2 \to \mathbb{C}_0$ *and a chosen object* $1 \in \mathbb{C}_0$ *with functions*

$$pair : \mathbb{C}(\Gamma; X) \times \mathbb{C}(\Gamma; Y) \to \mathbb{C}(\Gamma; X \times Y)$$

$$unit : 1 \to \mathbb{C}(\Gamma; 1)$$

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

$$pair(f, g) \circ \gamma \sqsupseteq\sqsubseteq pair(f \circ \gamma, g \circ \gamma)$$

$$unit \circ \gamma \sqsupseteq\sqsubseteq unit$$

and morphisms

$$\pi_1 : \mathbb{C}(X \times Y; X) \quad \pi_2 : \mathbb{C}(X \times Y; Y)$$

such that the function given by

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

is an inverse to pair up to $\sqsupseteq\sqsubseteq$*.*

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

4.2 Soundness and Completeness for Gradual Type Theory

Now we extend the soundness and completeness theorems for preorder type theory to full gradual type theory.

Definition 11 (GTT category)**.** *A GTT category is a cartesian closed VCP coreflective equipment with a greatest object and local bottoms.*

Definition 12 (Interpretation of Gradual Type Theory/Soundness)**.** *Let* $\mathbb{C}$ *be a GTT category and* $\Sigma = (\Sigma_0, \Sigma_1, \Sigma_2, \Sigma_3)$ *a GTT signature. Then interpreting all syntax as relative to* Σ*,*

1. *Given an interpretation* $(\!|\cdot|\!) : \Sigma_0 \to \mathbb{C}_0$*, we extend it to a compositional function* $[\![\cdot]\!] : GTT_0(\Sigma_0) \to \mathbb{C}_0$ *defined by:*

$$\begin{aligned}
[\![X \in \Sigma_0]\!] &= (\!|X|\!) \\
[\![?]\!] &= \top \\
[\![A \to B]\!] &= [\![A]\!] \to [\![B]\!] \\
[\![A \times B]\!] &= [\![A]\!] \times [\![B]\!] \\
[\![1]\!] &= 1
\end{aligned}$$

2. *If for every* $(A, B) \in \Sigma_1$ *then* $[\![A]\!] \sqsubseteq [\![B]\!]$ *holds, then for* any *derivation* $A \sqsubseteq B$*,* $[\![A]\!] \sqsubseteq [\![B]\!]$ *holds. For the dynamic type, this is by definition of a greatest element, and for the congruence rules for type constructors, by monotonicity of the closed and cartesian structures.*

3. *Given an interpretation of the function symbols* $(\!|\cdot|\!) : \Sigma_2(A_1, \ldots, A_n; B) \to \mathbb{C}_1([\![A_1]\!], \ldots, [\![A_n]\!]; [\![B]\!])$*, we extend it do a compositional semantics function* $[\![\cdot]\!] : \Gamma \vdash \cdot : A \to \mathbb{C}_1([\![\Gamma]\!], [\![A]\!])$ *by*

$$\begin{aligned}
[\![\Gamma, x : A, \Delta \vdash x : A]\!] &= x(\lfloor \Gamma \rfloor, A, \lfloor \Delta \rfloor) \\
[\![\Gamma \vdash f(\gamma(x_1), \ldots, \gamma(x_n))]\!] &= (\!|f|\!) \circ [\![\gamma]\!] \\
[\![\langle B \leftarrow\!\!\prec A \rangle t]\!] &= u_{[\![A]\!],[\![B]\!]} \circ [\![t]\!] \\
[\![\langle A \twoheadleftarrow B \rangle t]\!] &= d_{[\![A]\!],[\![B]\!]} \circ [\![t]\!] \\
[\![\mathcal{O}]\!] &= \bot \\
[\![\lambda x : A.t]\!] &= \lambda([\![t]\!]) \\
[\![tu]\!] &= app \circ ([\![t]\!], [\![u]\!]) \\
[\![(t_1, t_2)]\!] &= pair([\![t_1]\!], [\![t_2]\!]) \\
[\![\pi_i t]\!] &= \pi_i \circ [\![t]\!] \\
[\![()]\!] &= unit
\end{aligned}$$

4. *If for every* $(t, t') \in \Sigma_3$*,* $[\![t]\!] \sqsubseteq [\![t']\!]$ *holds, then for* any *derivation* $\Gamma \sqsubseteq \Gamma' \vdash t \sqsubseteq t' : A \sqsubseteq A'$*,* $[\![t]\!] \sqsubseteq [\![t']\!]$ *holds. For casts, this is by definition of an equipment, and for type errors, by definition of a local bottom. For type constructors, the congruence rules hold by monotonicity of the cartesian and closed structures, and the* $\beta\eta$ *by the equational laws for the closed structure.*

5. *An interpretation* $(\!|\cdot|\!) : \Sigma \to \mathbb{C}$ *is a pair of base type and function symbol interpretation functions satisfying the conditions of the definitions above.*

Theorem 3 (Completeness of GTT Category Semantics)**.** *For any GTT signature* Σ*,*

1. *for any* GTT_Σ *types* A, B *if for every interpretation* $(\!|\cdot|\!) : \Sigma \to \mathbb{C}$*,* $[\![A]\!] \sqsubseteq [\![B]\!]$ *holds, then* $A \sqsubseteq B$ *is derivable.*

2. *For any* GTT_Σ *contexts* $\Phi : \Gamma \sqsubseteq \Gamma'$*, types* $A \sqsubseteq A'$*, and terms* $\Gamma \vdash t : A$ *and* $\Gamma' \vdash t' : A'$*, if for every interpretation* $[\![t]\!] \sqsubseteq [\![t']\!]$*, then* $\Phi \vdash t \sqsubseteq t' : A \sqsubseteq A'$ *derivable.*

As usual, the proof of completeness is by building a GTT category from the syntax such that the true dynamism theorems are precisely the derivable ones.

Together these theorems imply that the syntax is initial: the semantics given by Definition 12 is the unique extension making a morphism of GTT categories using the GTT category structure of Theorem 3.

Theorem 4 (Initiality)**.** *Given a GTT interpretation* $(\!|\cdot|\!) : \Sigma \to \mathbb{C}$*, The semantics* $[\![\cdot]\!] : GTT_\Sigma \to \mathbb{C}$ *is the unique (up to* $\sqsupseteq\sqsubseteq$*)* morphism of GTT categories *extending it: it is a functor of preorder categories that preserves GTT structure up to* $\sqsupseteq\sqsubseteq$*.*

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). The contract construction then "implements" gradual typing using the morphisms of the non-gradual "programming language" $\mathbb{C}$.

Coreflections To build a GTT model from $\mathbb{C}$, we need to choose an interpretation of type dynamism (the ordering on objects of the VCP category) that induces appropriate casts, which we know by Theorem 1.6 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). We will use the term coreflection since it is shortest. Since type dynamism judgments must induce a coreflection, we will construct a model where the semantics of a type dynamism judgment $A \sqsubseteq B$ is literally 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: they are categories internal to the category of categories, rather than the category of preorders. Concretely, 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 13 (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, a slight variation on a theorem in [27]: We can think of this as a model of a "type dynamism proof-relevant" system where there might be many different ways that $A \sqsubseteq B$ (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*. Specifically, in the well-pointed case 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 [27]

Definition 14 (Equipment of Coreflections [27])**.** *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 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 equivalence:* $d \circ u \equiv id$*. 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, any mixed-variance functor preserves coreflections [36, 31], 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 (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* $\bot$*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.

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 $A \sqsubseteq B$, but there may be different coreflections from A to B. It turns out that we can solve both problems at once by taking what we call the "vertical slice" category [3]over an object $D \in \text{CoReflect}(\mathbb{C})$ 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 $(u_A, d_A) : A \triangleleft D$.[4] Thus, gradual types are modeled as coreflections into the dynamic type, analogous to Scott's "retracts of a universal domain" [26]. 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($\mathbb{C}$)$/D$. Further, the identity coreflection $(\text{id}, \text{id}) : D \triangleleft D$ is a vertically greatest element since any morphism is factorized by the identity. This interpretation also helps us understand the unusual transitivity rule mentioned in Section 2.1:

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

Consider the interpretation of this rule where we have (unique) morphisms $(l_0, r_0) : A \triangleleft A', (l_1, r_1) : A' \triangleleft A'', (l_2, r_2) : A \triangleleft A''$. Then in the bottom we are given x, x'' with $l_2(x) \sqsubseteq x''$ but in the premises we need to produce a x' with $l_1(x) \sqsubseteq x'$ and $l_2(x') \sqsubseteq x''$. However by the uniqueness of factorizations, we also know $l_2 = l_1 \circ l_0$ so we can define $x' = l_1(x)$.

Definition 15 (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* $\mathbb{E}$.

[3]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.

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

Next consider 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 theorems:

Theorem 6 (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 $\mathbb{C}/D$*

Summary Finally, we construct a *virtually* cartesian model from a cartesian model :

Definition 16 (VCP Category from a Cartesian Preorder Category)**.** *If $\mathbb{C}$ is a cartesian preorder category, then we can construct a VCP category Virt($\mathbb{C}$) by*

1. *Virt*$(\mathbb{C})_0 = \mathbb{C}_0$
2. *Virt*$(\mathbb{C})(A_1, \ldots, A_n; B) = \mathbb{C}_1(A_1 \times (\cdots (A_n \times 1) \cdots); B)$

Proof. This follows from a quite general result of [5]. □

Combining these constructions, we produce:

Theorem 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 $c_\to : (D \to D) \triangleleft D$, $c_\times : (D \times D) \triangleleft D$, and $c_1 : 1 \triangleleft D$, then ($Virt(CoReflect(\mathbb{C})/id_d), c_\to, c_\times, c_1$) 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.

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 $PreOrd_\bot$ whose objects are preorders with a least element, which following domain-theoretic terminology we call "pointed" preorders, and whose morphism 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 the least solution of 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 $\bot$, which is 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.

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 [26]. 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 [36, 31, 24]) 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 $(D \times D) \lhd D$ and $(D \to D) \lhd D$.

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:

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

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 $\bot$. Then $\langle (? \times ?) \twoheadleftarrow ? \rangle \langle ? \hookleftarrow (? \to ?) \rangle x \not\equiv \mathfrak{O}$ because there are elements of the domain that are non-trivial both in the $D \times D$ position and $D \to D$ position.

Then we can construct a model of gradual typing using Theorem 7 with the 2-category of pointed domain preorders, monotone continuous functions and whose 2-cells are given by the *error ordering*.

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 $\mathfrak{O}$ that models the dynamic type error. These can be described as preorders internal to the category of domains.

Definition 17 (Domain Preorders)**.** *1. A pre-domain preorder is a set X with two orderings $\leq$ and $\sqsubseteq$ such that $(X, \leq)$ is an ω-complete partial order and $\sqsubseteq$ is a preorder closed under limits of $\leq$-ω-chains.*

A continuous function of pre-domain preorders is a function of the underlying sets that is continuous with respect to $\leq$ and monotone with respect to $\sqsubseteq$.

2. A domain preorder is a pre-domain preorder with a $\leq$-least element $\bot$.

3. A pointed domain preorder is a domain preorder with a $\sqsubseteq$-least element $\mathfrak{O}$.

We want to model our types as pointed domain preorders because they have an interpretation for both divergence $\bot$ and type error $\mathfrak{O}$. The category of pointed domain preorders is an O-category in the sense of [36, 31], with all ω^{op} limits and so we can solve recursive domain equations there. We can define the wedge sum $A \oplus B$ to be the disjoint union quotiented by identifying $\bot_A = \bot_B$ and $\mathfrak{O}_A = \mathfrak{O}_B$. Furthermore it is cartesian closed and unit, product, exponential and $\oplus$ are all locally continuous mixed-variance functors. We can then construct a suitable dynamic type in the same fashion as for domains:

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

Then we can construct coreflections $D \times D \lhd D$ and $D \to D \lhd D$ in the same way as the model for preorders: the downcast produces $\mathfrak{O}$ unless it is the $D \times D$ (respectively $D \to D$) case and the $1 \lhd D$ is the unique coreflection between those objects.

7 Related and Future Work

Logic and Semantics of Dynamism Our logic and semantics of type and term dynamism builds on the formulation introduced with the gradual guarantee in [28], but the rules of our system differ in several ways. First, 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. Second, our system also includes the β, η equivalences as equi-dynamism axioms, making term dynamism more semantic.

As a relational logic with a sound and complete categorical semantics, it has commonalities with logics for parametric polymorphism [25], and the categorical semantics in terms of *reflexive graph categories* which are like double categories where vertical arrows lack composition [23]. 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 [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.

Contracts as Coreflections Our semantic model of contracts as coreflections has precedent in much previous work, though we are the first to identify 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 work denotational work use coreflections to solve mixed-variance domain equations [26, 36, 31]. 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 [24]'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 the 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 [4].

Recent work on interoperability in a (non-gradual) dependently typed language [6, 7] defines casts as "partial type equivalences" between types, which are defined as a pair of terms $f : A \to B$, $g : B \to A$ satisfying projection in *both* directions: $f \circ g \sqsubseteq \mathrm{id}$ and $g \circ f \sqsubseteq \mathrm{id}$. This does not model type dynamism, but rather the notion of "general" cast that is not necessarily an upcast or a downcast. Using our decomposition of general casts into an upcast followed by a downcast, we can prove in our logic that any general cast is a

partial type equivalence. Their work also identifies Galois connections/insertions as being a possible model of upcasts and downcasts, but they do not develop the idea further.

"Dogma" of Gradual Typing There are two recent proposals for a more general theory of gradual typing: Abstracting Gradual Typing (AGT) [12] and the Gradualizer [3]. 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. Finally, AGT is based on abstract interpretation and uses a Galois insertion between gradual types and sets of static types, but we do not see a precise relationship to our use of coreflections.

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 on triple casts in [30], 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.

Future Work In this paper we have shown that the combination of soundness and graduality produces strong specifications for call-by-name gradual typing implementations. However so far we have only validated this by denotational semantics, and we plan to develop *operational models* of this kind of gradual type theory where program equivalence is given by contextual equivalence and term dynamism is modeled by a type of contextual approximation. We also will investigate extensions to richer languages. First, we would like to develop a similar theory for call-by-value gradual typing, as every gradually typed language in use today is call-by-value, and to extend call-by-name with an appropriate notion of positive type (booleans, general sums). We plan to build on existing work on categorical semantics and universal properties of types in call-by-value [19, 32]. The combination of gradual typing and parametric polymorphism has proven quite complex [20, 22, 1, 16]. 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.

Acknowledgments We thank Amal Ahmed for the countless insightful discussions of this work.

References

[1] A. Ahmed, D. Jamner, J. G. Siek, and P. Wadler. Theorems for free for free: Parametricity, with and without types. In *International Conference on Functional Programming (ICFP)*, 2017.

[2] F. Bañados Schwerter, R. Garcia, and E. 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.

[3] M. Cimini and J. 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.

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

[5] G. S. H. Cruttwell and M. A. Shulman. A unified framework for generalized multicategories. *Theory and Applications of Categories*, 24(21), 2009.

[6] P.-E. Dagand, N. Tabareau, and E. Tanter. Partial type equivalences for verified dependent interoperability. In *International Conference on Functional Programming (ICFP)*, 2016.

[7] P.-E. Dagand, N. Tabareau, and É. Tanter. Foundations of Dependent Interoperability. working paper or preprint, 2017.

[8] B. P. Dunphy. *Parametricity As a Notion of Uniformity in Reflexive Graphs*. PhD thesis, Champaign, IL, USA, 2002.

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

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

[11] R. B. Findler and M. Felleisen. Contracts for higher-order functions. In *International Conference on Functional Programming (ICFP)*, pages 48–59, Sept. 2002.

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

[13] N. Ghani, P. Johann, F. N. Forsberg, F. Orsanigo, and T. Revell. Bifibrational functorial semantics for parametric polymorphism. In *Proceedings of Mathematical Foundations of Program Semantics*, 2015.

[14] F. Henglein. Dynamic typing: Syntax and proof theory. *Sci. Comput. Programming*, 22(3):197–230, 1994.

[15] A. Igarashi, P. Thiemann, V. Vasconcelos, and P. Wadler. Gradual session types. In *International Conference on Functional Programming (ICFP)*, 2017.

[16] Y. Igarashi, T. Sekiyama, and A. 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] N. Lehmann and E. Tanter. Gradual refinement types. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages*, POPL 2017.

[19] P. B. Levy. *Call-by-Push-Value*. Ph. D. dissertation, Queen Mary, University of London, London, UK, Mar. 2001.

[20] J. Matthews and A. Ahmed. Parametric polymorphism through run-time sealing, or, theorems for low, low prices! In *European Symposium on Programming (ESOP)*, Mar. 2008.

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

[22] G. Neis, D. Dreyer, and A. Rossberg. Non-parametric parametricity. In *International Conference on Functional Programming (ICFP)*, pages 135–148, Sept. 2009.

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

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

[25] G. Plotkin and M. Abadi. A logic for parametric polymorphism. *Typed Lambda Calculi and Applications*, pages 361–375, 1993.

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

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

[28] J. Siek, M. Vitousek, M. Cimini, and J. T. Boyland. Refined criteria for gradual typing. In *1st Summit on Advances in Programming Languages*, SNAPL 2015, 2015.

[29] J. G. Siek and W. Taha. Gradual typing for functional languages. In *Scheme and Functional Programming Workshop (Scheme)*, pages 81–92, Sept. 2006.

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

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

[32] S. Staton and P. B. Levy. Universal properties of impure programming languages. In *ACM Symposium on Principles of Programming Languages (POPL)*, 2013.

[33] S. Tobin-Hochstadt and M. Felleisen. Interlanguage migration: From scripts to programs. In *Dynamic Languages Symposium (DLS)*, pages 964–974, Oct. 2006.

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

[35] P. Wadler and R. B. Findler. Well-typed programs can't be blamed. In *European Symposium on Programming (ESOP)*, pages 1–16, Mar. 2009.

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

[37] R. Wolff, R. Garcia, E. Tanter, and J. Aldrich. Gradual typestate. In *Proceedings of the 25th European Conference on Object-oriented Programming*, ECOOP'11, 2011.