The Combination of Dynamic and Static Typing from a Categorical Perspective

Harley Eades III[1,2]

Computer and Information Sciences
Augusta University
Augusta, USA

Michael Townsend[3]

Computer and Information Sciences
Augusta University
Augusta, USA

Abstract

This paper is part one of a two part series. In this paper we introduce a new categorical model based on retracts that combines static and dynamic typing. This model is initially based on the seminal work of Scott who showed that the untyped λ-calculus can be considered as typed using retracts. We then show that our model gives rise to a new and simple type system which combines static and dynamic typing. We then show that Siek and Taha's gradually typed λ-calculus can be interpreted in our new model. Finally, we set the stage for a followup paper that greatly extends this type system with bounded quantification and lists, and then develops a gradually typed surface language that uses our new type system as a core casting calculus.

Keywords: static typing, dynamic typing, gradual typing, categorical semantics, retract,typed lambda-calculus, untyped lambda-calculus, functional programming

1 Introduction

Scott [7] showed how to model the untyped λ-calculus within a cartesian closed category, $\mathcal{C}$, with a distinguished object we will call ? – read as the type of untyped terms, or the unknown type – such that the object [4] $? \to ?$ is a retract of ?. That is, there are morphisms $\mathsf{squash} : (? \to ?) \longrightarrow ?$ and $\mathsf{split} : ? \longrightarrow (? \to ?)$ where $\mathsf{squash}; \mathsf{split} = \mathsf{id} : (? \to ?) \longrightarrow (? \to ?)$ [5]. For example, taking these mor-

[1] Thanks: TODO

[2] Email:heades@augusta.edu

[3] Email: mitownsend@augusta.edu

[4] We will use the terms "object" and "type" interchangeably.

[5] We denote composition of morphisms by $f; g : A \longrightarrow C$ given morphisms $f : A \longrightarrow B$ and $g : B \longrightarrow C$.

phisms as terms in the typed λ-calculus we can define the prototypical looping term $(\lambda x.x\,x)(\lambda x.x\,x)$ by $(\lambda x : ?.(\mathsf{split}\,x)\,x)\,(\mathsf{squash}\,(\lambda x : ?.(\mathsf{split}\,x)\,x))$.

In the same volume as Scott, Lambek [5] showed that cartesian closed categories also model the typed λ-calculus. However, Scott's model required a retract, but Lambek's does not. Now combine both of Scott and Lambek's work by adding to $\mathcal{C}$ the type of untyped terms ?, squash, and split. At this point $\mathcal{C}$ is a model of both the typed and the untyped λ-calculus. However, the two theories are really just sitting side by side in $\mathcal{C}$ and cannot really interact much.

Suppose A is an atomic type. Then we add to $\mathcal{C}$ the morphisms $\mathsf{box} : A \longrightarrow ?$ and $\mathsf{unbox} : ? \longrightarrow A$ such that $\mathsf{box}; \mathsf{unbox} = \mathsf{id} : A \longrightarrow A$ making A a retract of ?. This is the bridge allowing the typed world to interact with the untyped one. We can think of box as injecting typed data into the untyped world, and unbox as taking it back. Notice that the only time we can actually get the typed data back out is if it were injected into the untyped world initially. In the model this is enforced through composition, but in the language this will be enforced at runtime, and hence, requires the language to contain dynamic typing. Thus, what we have just built up is a categorical model that offers a new perspective of how to combine static and dynamic typing.

Siek and Taha [8] define gradual typing to be the combination of both static and dynamic typing that allows for the programmer to program in dynamic style without the need for the them to explicitly insert casts into their programs. Siek and Taha's initial paper laid out the first gradually typed λ-calculus, but Siek et al. [9] layout a refined criteria for what metatheortic properties a gradual type system should have called the gradual guarantee. In this paper we show that our categorical model leads to a new core casting calculus for gradual type systems.

Siek and Taha's gradually typed λ-calculus is defined as the simply typed λ-calculus with the type of untyped terms ? and the following new application rule:

$$\frac{\Gamma \vdash_{\mathsf{S}} t_1 : C \qquad \Gamma \vdash_{\mathsf{S}} t_2 : A_2 \quad A_2 \sim A_1 \quad \mathsf{fun}(C) = A_1 \to B_1}{\Gamma \vdash_{\mathsf{S}} t_1\; t_2 : B_1}\;{\to_e}$$

The premise $A_2 \sim A_1$ is read, the type A_2 is consistent with the type A_1, and is defined in Figure 1. If we squint we can see split, squash, box, and unbox hiding in the definition of the previous rules, but they have been suppressed. We will show that when one uses this typing rule then one is really implicitly using a casting morphism built from split, squash, box, and unbox. In fact, the consistency relation $A \sim B$ can be interpreted as such a morphism. Then the typing above can be read semantically as a saying if a casting morphism exists, then the type really can be converted into the necessary type. The premise $\mathsf{fun}(C) = A_1 \to B_1$ requires that C either be $A_1 \to B_1$ or $C = ?$ and $\mathsf{fun}(C) = ? \to ?$ where $A_1 = ?$ and $B_1 = ?$. Again, we can see split and box hiding in the shadows. When $C = ?$, then $A_1 = ?$, and hence, $A_2 \sim A_1$ will correspond to box, and the premise $\mathsf{fun}(C) = ? \to ?$ corresponds to using split. Thus, in this case the term $t_1\; t_2$ can be converted into a term in our calculus by $(\mathsf{split}_{(?\to?)}\,t_1)\,(\mathsf{box}_{A_2}\,t_2)$. An interesting point about this rule is that dynamically typed programs tend toward statically typed programs. However, the gradual guarantee shows that any program in the gradual type system can slide

either more towards static typing or more towards dynamic typing by inserting or removing casts.

Contributions. This paper offers the following contributions:

- A new categorical model for gradual typing for functional languages. We show how to interpret Siek and Taha's [9] gradual type system in the categorical model outlined above.
- We then extract a functional programming language called Simply Typed Grady – or just Grady – from the categorical model via the Curry-Howard-Lambek correspondence. This is not a gradual type system, but can be seen as an alternative core casting calculus in which Siek and Taha's gradual type system can be translated to.

Related work. We now give a brief summary of related work. Each of the articles discussed below can be consulted for further references.

- Abadi et al. [1] combine dynamic and static typing by adding a new type called Dynamic along with a new case construct for pattern matching on types. We do not add such a case construct, and as a result, show that we can obtain a surprising amount of expressivity without it. They also provide denotational models.
- Henglein [4] defines the dynamic λ-calculus by adding a new type Dyn to the simply typed λ-calculus and then adding primitive casting operations called tagging and check-and-untag. These new operations tag type constructors with their types. Then untagging checks to make sure the target tag matches the source tag, and if not, returns a dynamic type error. These operations can be used to build casting coercions which are very similar to our casting morphisms. We can also define split, squash, box, and unbox in terms of Henglein's casting coercions. However, our setting is a bit more strict then his, because his "boxing" and "unboxing" operations form an isomorphism, but we show that a retract is only needed.
- As we mentioned in the introduction Siek and Taha [8] were the first to define gradual typing especially for functional languages. We show that their language can be given a straightforward categorical model in cartesian closed categories. Since their original paper introducing gradual types lots of languages have adopted it, but the term "gradual typing" started to become a catch all phrase for any language combining dynamic and static typing. As a result of this Siek et al. [9] later refine what it means for a language to support gradual typing by specifying the necessary metatheoretic properties a gradual type system must satisfy called the gradual guarantee.
- The categorical model we give here is with respect to the core casting calculus. In future work we plan to investigate a categorical model for the surface language, and we believe that Garcia's [3] work showing how to extract the gradual type system from a fully static type system using abstract interpretation will play a key role. Abstract interpretation uses Galois connections which can be studied as adjoints in the category of posets. Garcia shows that one can start with a static type system, and then use abstract interpretation

to infer the gradually typed surface language complete with both statics and dynamics.

2 The Categorical Model

The model we develop here builds on the seminal work of Lambek [5] and Scott [7]. Lambek [5] showed that the typed λ-calculus can be modeled by a cartesian closed category. In the same volume as Lambek, Scott essentially showed that the untyped λ-calculus is actually typed. That is, typed theories are more fundamental than untyped ones. He accomplished this by adding a single type, ?, and two functions $\mathsf{squash} : (? \to ?) \to ?$ and $\mathsf{split} : ? \to (? \to ?)$, such that, $\mathsf{squash}; \mathsf{split} = \mathsf{id} : (? \to ?) \to (? \to ?)$. At this point he was able to translate the untyped λ-calculus into this unityped one. Categorically, he modeled split and squash as the morphisms in a retract within a cartesian closed category – the same model as typed λ-calculus.

Definition 2.1 Suppose $\mathcal{C}$ is a category. Then an object A is a **retract** of an object B if there are morphisms $i : A \longrightarrow B$ and $r : B \longrightarrow A$ such that $i; r = \mathsf{id}_A$.

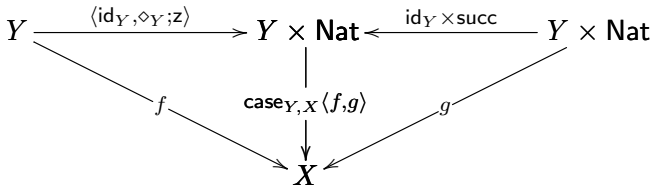
Thus, $? \to ?$ is a retract of ?, but we also require that $? \times ?$ be a retract of ?; this is not new, see Lambek and Scott [6]. Putting this together we obtain Scott's model of the untyped λ-calculus.

Definition 2.2 An **untyped λ-model**, $(\mathcal{C}, ?, \mathsf{split}, \mathsf{squash})$, is a cartesian closed category $\mathcal{C}$ with a distinguished object ? and morphisms $\mathsf{squash} : S \longrightarrow ?$ and $\mathsf{split} : ? \longrightarrow S$ making the object S a retract of ?, where S is either $? \to ?$ or $? \times ?$.

Theorem 2.3 (Scott [7]) *An untyped λ-model is a sound and complete model of the untyped λ-calculus.*

We now show how to model the natural numbers with their (non-recursive) eliminator using what we call a non-recursive natural number object. This is a simplification a traditional natural number object; see Lambek and Scott [6] for an introduction to natural number objects.

Definition 2.4 Suppose $\mathcal{C}$ is a cartesian closed category. A **non-recursive natural number object (NRNO)** is an object Nat of $\mathcal{C}$ and morphisms $\mathsf{z} : 1 \longrightarrow \mathsf{Nat}$ and $\mathsf{succ} : \mathsf{Nat} \longrightarrow \mathsf{Nat}$ of $\mathcal{C}$, such that, for any morphisms $f : Y \longrightarrow X$ and $g : Y \times \mathsf{Nat} \longrightarrow X$ of $\mathcal{C}$ there is a unique morphism $\mathsf{case}_X \langle f, g \rangle : Y \times \mathsf{Nat} \longrightarrow X$ such that the following diagrams commute:



Informally, the two diagrams essentially assert that we can define case_X as follows:

$$\mathsf{case}_{Y,X} \langle f, g \rangle \, y \, 0 = f \, y \qquad \mathsf{case}_{Y,X} \langle f, g \rangle \, y \, (\mathsf{succ} \, n) = g \, y \, n$$

So far we can model the untyped and the typed λ-calculi within a cartesian closed category, but we do not have any way of moving typed data into the untyped part and vice versa. To accomplish this we add two new morphisms $\mathsf{box}_C : C \longrightarrow ?$ and $\mathsf{unbox}_C : ? \longrightarrow C$ such that each atomic type, C, is a retract of ?. This enforces that the only time we can really consider something as typed is if it were boxed up in the first place.

Definition 2.5 A **gradual λ-model**, $(\mathcal{T}, \mathcal{C}, ?, \mathsf{T}, \mathsf{split}, \mathsf{squash}, \mathsf{box}, \mathsf{unbox})$, where $\mathcal{T}$ is a discrete category with at least two objects Nat and Unit, $\mathcal{C}$ is a cartesian closed category with a NRNO, $(\mathcal{C}, ?, \mathsf{split}, \mathsf{squash})$ is an untyped λ-model, $\mathsf{T} : \mathcal{T} \longrightarrow \mathcal{C}$ is an embedding – a full and faithful functor that is injective on objects – and for every object A of $\mathcal{T}$ there are morphisms $\mathsf{box}_A : TA \longrightarrow ?$ and $\mathsf{unbox}_A : ? \longrightarrow TA$ making TA a retract of ?.

We call the category $\mathcal{T}$ the category of atomic types. We call an object, A, **atomic** iff there is some object A' in $\mathcal{T}$ such that $A = \mathsf{T}A'$. Note that we do not consider ? an atomic type. The model really is the cartesian closed category $\mathcal{C}$, but it is extended with the structure of both the typed and the untyped λ-calculus with the ability to cast data.

As the model is defined it is unclear if we can cast any type to ?, and vice versa, but we must be able to do this in order to model full dynamic typing. In the remainder of this section we show that we can build up such casts in terms of the basic features of our model.

We call any morphism defined completely in terms of id, the functors $- \times -$ and $- \to -$, split and squash, and box and unbox a **casting morphism**. To cast any type A to ? we will build casting morphisms that first take the object A to its skeleton, and then takes the skeleton to ?.

Definition 2.6 Suppose $(\mathcal{T}, \mathcal{C}, ?, \mathsf{T}, \mathsf{split}, \mathsf{squash}, \mathsf{box}, \mathsf{unbox})$ is a gradual λ-model. Then the **skeleton** of an object A of $\mathcal{C}$ is an object S that is constructed by replacing each atomic type in A with ?. Given an object A we denote its skeleton by $\mathsf{skeleton}\ A$.

One should think of the skeleton of an object as the supporting type structure of the object, but we do not know what kind of data is actually in the structure. For example, the skeleton of the object Nat is ?, and the skeleton of $(\mathsf{Nat} \times \mathsf{Unit}) \to \mathsf{Nat} \to \mathsf{Nat}$ is $(? \times ?) \to ? \to ?$.

The next definition defines a means of constructing a casting morphism that casts a type A to its skeleton and vice versa. This definition is by mutual recursion on the input type.

Definition 2.7 Suppose $(\mathcal{T}, \mathcal{C}, ?, \mathsf{T}, \mathsf{split}, \mathsf{squash}, \mathsf{box}, \mathsf{unbox})$ is a gradual λ-model. Then for any object A whose skeleton is S we define the morphisms $\widehat{\mathsf{box}}_A : A \longrightarrow S$

and $\widehat{\mathsf{unbox}}_A : S \longrightarrow A$ by mutual recursion on A as follows:

$$\begin{array}{l|l}
\widehat{\mathsf{box}}_A = \mathsf{box}_A & \widehat{\mathsf{unbox}}_A = \mathsf{unbox}_A \\
\quad \text{when } A \text{ is atomic} & \quad \text{when } A \text{ is atomic} \\
\widehat{\mathsf{box}}_? = \mathsf{id}_? & \widehat{\mathsf{unbox}}_? = \mathsf{id}_? \\
\widehat{\mathsf{box}}_{(A_1 \to A_2)} = \widehat{\mathsf{unbox}}_{A_1} \to \widehat{\mathsf{box}}_{A_2} & \widehat{\mathsf{unbox}}_{(A_1 \to A_2)} = \widehat{\mathsf{box}}_{A_1} \to \widehat{\mathsf{unbox}}_{A_2} \\
\widehat{\mathsf{box}}_{(A_1 \times A_2)} = \widehat{\mathsf{box}}_{A_1} \times \widehat{\mathsf{box}}_{A_2} & \widehat{\mathsf{unbox}}_{(A_1 \times A_2)} = \widehat{\mathsf{unbox}}_{A_1} \times \widehat{\mathsf{unbox}}_{A_2}
\end{array}$$

The definition of both $\widehat{\mathsf{box}}$ or $\widehat{\mathsf{unbox}}$ uses the functor $- \to - : \mathcal{C}^{\mathsf{op}} \times \mathcal{C} \longrightarrow \mathcal{C}$ which is contravariant in its first argument, and thus, in that contravariant position we must make a recursive call to the opposite function, and hence, they must be mutually defined. Every call to either $\widehat{\mathsf{box}}$ or $\widehat{\mathsf{unbox}}$ in the previous definition is on a smaller object than the input object. Thus, their definitions are well founded. Furthermore, $\widehat{\mathsf{box}}$ and $\widehat{\mathsf{unbox}}$ form a retract between A and S.

Lemma 2.8 (Boxing and Unboxing Lifted Retract) *Suppose* $(\mathcal{T}, \mathcal{C}, ?, \mathsf{T}, \mathsf{split}, \mathsf{squash}, \mathsf{box}, \mathsf{unbox})$ *is a gradual* λ*-model. Then for any object* A, $\widehat{\mathsf{box}}_A ; \widehat{\mathsf{unbox}}_A = \mathsf{id}_A : A \longrightarrow A$.

Proof. This proof holds by induction on the form A. Please see Appendix A.1 for the complete proof. □

As an example, suppose we wanted to cast the type $(\mathsf{Nat} \times ?) \to \mathsf{Nat}$ to its skeleton $(? \times ?) \to ?$. Then we can obtain a casting morphisms that will do this as follows:

$$\widehat{\mathsf{box}}_{((\mathsf{Nat} \times ?) \to \mathsf{Nat})} = (\mathsf{unbox}_{\mathsf{Nat}} \times \mathsf{id}_?) \to \mathsf{box}_{\mathsf{Nat}}$$

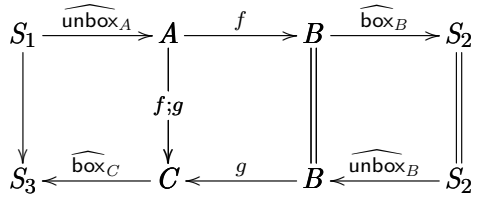
We can also cast a morphism $A \xrightarrow{f} B$ to a morphism

$$S_1 \xrightarrow{\widehat{\mathsf{unbox}}_A} A \xrightarrow{f} B \xrightarrow{\widehat{\mathsf{box}}_B} S_2$$

where $S_1 = \mathsf{skeleton}\ A$ and $S_2 = \mathsf{skeleton}\ B$. Now if we have a second

$$S_2 \xrightarrow{\widehat{\mathsf{unbox}}_B} B \xrightarrow{g} C \xrightarrow{\widehat{\mathsf{box}}_C} S_3$$

then their composition reduces to composition at the typed level:



The right most diagram commutes because B is a retract of S_2, and the left unannotated arrow is the composition $\widehat{\mathsf{unbox}}_A ; f ; g ; \widehat{\mathsf{box}}_C$. This tells us that we have a

functor $\mathsf{S} : \mathcal{C} \longrightarrow \mathcal{S}$:

$$\mathsf{S}A = \mathsf{skeleton}\ A$$
$$\mathsf{S}(f : A \longrightarrow B) = \widehat{\mathsf{unbox}}_A; f; \widehat{\mathsf{box}}_A$$

where $\mathcal{S}$ is the full subcategory of $\mathcal{C}$ consisting of the skeletons and morphisms between them, that is, $\mathcal{S}$ is a cartesian closed category with one basic object ? such that $(\mathcal{S}, ?, \mathsf{split}, \mathsf{squash})$ is an untyped λ-model. The following turns out to be true.

Lemma 2.9 (S is faithful) *Suppose $(\mathcal{T}, \mathcal{C}, ?, \mathsf{T}, \mathsf{split}, \mathsf{squash}, \mathsf{box}, \mathsf{unbox})$ is a gradual λ-model, and $(\mathcal{S}, ?, \mathsf{split}, \mathsf{squash})$ is the category of skeletons. Then the functor $\mathsf{S} : \mathcal{C} \longrightarrow \mathcal{S}$ is faithful.*

Proof. This proof follows from the definition S and Lemma 2.8. For the full proof see Appendix A.2. □

Thus, we can think of the functor S as an injection of the typed world into the untyped one.

Now that we can cast any type into its skeleton we must show that every skeleton can be cast to ?. We do this similarly to the above and lift split and squash to arbitrary skeletons.

Definition 2.10 Suppose $(\mathcal{S}, ?, \mathsf{split}, \mathsf{squash})$ is the category of skeletons. Then for any skeleton S we define the morphisms $\widehat{\mathsf{squash}}_S : S \longrightarrow ?$ and $\widehat{\mathsf{split}}_S : ? \longrightarrow S$ by mutual recursion on S as follows:

$$\begin{aligned}
&\widehat{\mathsf{squash}}_? = \mathsf{id}_? \\
&\widehat{\mathsf{squash}}_{(S_1 \to S_2)} = (\widehat{\mathsf{split}}_{S_1} \to \widehat{\mathsf{squash}}_{S_2}); \mathsf{squash}_{? \to ?} \\
&\widehat{\mathsf{squash}}_{(S_1 \times S_2)} = (\widehat{\mathsf{squash}}_{S_1} \times \widehat{\mathsf{squash}}_{S_2}); \mathsf{squash}_{? \times ?} \\
\hline
&\widehat{\mathsf{split}}_? = \mathsf{id}_? \\
&\widehat{\mathsf{split}}_{(S_1 \to S_2)} = \mathsf{split}_{? \to ?}; (\widehat{\mathsf{squash}}_{S_1} \to \widehat{\mathsf{split}}_{S_2}) \\
&\widehat{\mathsf{split}}_{(S_1 \times S_2)} = \mathsf{split}_{? \times ?}; (\widehat{\mathsf{split}}_{S_1} \times \widehat{\mathsf{split}}_{S_2})
\end{aligned}$$

As an example we will construct the casting morphism that casts the skeleton $(? \times ?) \to ?$ to ?: $\widehat{\mathsf{squash}}_{(? \times ?) \to ?} = (\mathsf{split}_{? \times ?} \to \mathsf{id}_?); \mathsf{squash}_{? \to ?}$.

Lemma 2.11 (Splitting and Squashing Lifted Retract) *Suppose $(\mathcal{S}, ?, \mathsf{split}, \mathsf{squash})$ is the category of skeletons. Then for any skeleton S,*

$$\widehat{\mathsf{squash}}_S; \widehat{\mathsf{split}}_S = \mathsf{id}_S : S \longrightarrow S$$

Proof. The proof is similar to the proof of the boxing and unboxing lifted retract (Lemma 2.8). □

There is also a faithful functor from $\mathcal{S}$ to $\mathcal{U}$ where $\mathcal{U}$ is the full subcategory of

$\mathcal{S}$ that consists of the single object ? and all its morphisms between it:

$$\begin{aligned}\mathsf{U}S &= ?\\ \mathsf{U}(f : S_1 \longrightarrow S_2) &= \widehat{\mathsf{split}}_{S_1}; f; \widehat{\mathsf{squash}}_{S_2}\end{aligned}$$

This finally implies that there is a functor $\mathsf{C} : \mathcal{C} \longrightarrow \mathcal{U}$ that injects all of $\mathcal{C}$ into the object ?.

Lemma 2.12 (Casting to ?) *Suppose* $(\mathcal{T}, \mathcal{C}, ?, \mathsf{T}, \mathsf{split}, \mathsf{squash}, \mathsf{box}, \mathsf{unbox})$ *is a gradual* λ*-model,* $(\mathcal{S}, ?, \mathsf{split}, \mathsf{squash})$ *is the full subcategory of skeletons, and* $(\mathcal{U}, ?)$ *is the full subcategory containing only* ? *and its morphisms. Then there is a faithful functor* $\mathsf{C} = \mathcal{C} \xrightarrow{\mathsf{S}} \mathcal{S} \xrightarrow{\mathsf{U}} \mathcal{U}$.

In a way we can think of $\mathsf{C} : \mathcal{C} \longrightarrow \mathcal{U}$ as a forgetful functor. It forgets the type information.

Getting back the typed information is harder. There is no nice functor from $\mathcal{U}$ to $\mathcal{C}$, because we need more information. However, given a type A we can always obtain a casting morphism from ? to A by $(\widehat{\mathsf{split}}_{(\mathsf{skeleton}\ A)}); (\widehat{\mathsf{unbox}}_A) : ? \longrightarrow A$. Finally, we have the following result.

Lemma 2.13 (Casting Morphisms to ?) *Suppose* $(\mathcal{T}, \mathcal{C}, ?, \mathsf{T}, \mathsf{split}, \mathsf{squash}, \mathsf{box}, \mathsf{unbox})$ *is a gradual* λ*-model, and* A *is an object of* $\mathcal{C}$*. Then there exists casting morphisms from* A *to* ? *and vice versa that make* A *a retract of* ?.

Proof. The two morphisms are as follows:

$$\begin{aligned}\mathsf{Box}_A &:= \widehat{\mathsf{box}}_A; \widehat{\mathsf{squash}}_{(\mathsf{skeleton}\ A)} : A \longrightarrow ?\\ \mathsf{Unbox}_A &:= \widehat{\mathsf{split}}_{(\mathsf{skeleton}\ A)}; \widehat{\mathsf{unbox}}_A : ? \longrightarrow A\end{aligned}$$

The fact the these form a retract between A and ? holds by Lemma 2.8 and Lemma 2.11. □

3 Gradual Typing

In this section we introduce a slight variation of the gradually typed functional language given by Siek et al. [9]. The syntax and typing rules of the gradual type system $\lambda^?_\to$ are defined in Figure 1. The main changes of the version of $\lambda^?_\to$ defined here from the original is that products and natural numbers have been added. The definition of products follows how casting is done for functions. So it allows casting projections of products, for example, it is reasonable for terms like $\lambda x : (? \times ?).(\mathsf{succ}\,(\mathsf{fst}\,x))$ to type check.

The typing rules depend on the following partial functions:

$$\begin{aligned}\mathsf{prod}(?) &= ? \times ? & \mathsf{fun}(?) &= ? \to ?\\ \mathsf{prod}(A_1 \times A_2) &= A_1 \times A_2 & \mathsf{fun}(A_1 \to A_2) &= A_1 \to A_2\end{aligned}$$

Syntax:

$$\begin{array}{rrcl} \text{(types)} & A, B, C, D & ::= & \mathsf{Unit} \mid \mathsf{Nat} \mid ? \mid A \times B \mid A_1 \to A_2 \\ \text{(terms)} & t & ::= & x \mid \mathsf{triv} \mid 0 \mid \mathsf{succ}\, t \mid \lambda x : A.t \mid t_1\, t_2 \mid (t_1, t_2) \mid \mathsf{fst}\, t \mid \mathsf{snd}\, t \\ \text{(contexts)} & \Gamma & ::= & \cdot \mid x : A \mid \Gamma_1, \Gamma_2 \end{array}$$

Typing Rules:

$$\frac{x : A \in \Gamma}{\Gamma \vdash_{\mathsf{S}} x : A}\ \mathrm{var} \qquad \frac{}{\Gamma \vdash_{\mathsf{S}} \mathsf{triv} : \mathsf{Unit}}\ \mathrm{unit} \qquad \frac{}{\Gamma \vdash_{\mathsf{S}} 0 : \mathsf{Nat}}\ \mathrm{zero}$$

$$\frac{\Gamma \vdash_{\mathsf{S}} t : A \quad \mathsf{nat}(A) = \mathsf{Nat}}{\Gamma \vdash_{\mathsf{S}} \mathsf{succ}\, t : \mathsf{Nat}}\ \mathrm{succ} \qquad \frac{\Gamma \vdash_{\mathsf{S}} t_1 : A_1 \quad \Gamma \vdash_{\mathsf{S}} t_2 : A_2}{\Gamma \vdash_{\mathsf{S}} (t_1, t_2) : A_1 \times A_2}\ \times$$

$$\frac{\Gamma \vdash_{\mathsf{S}} t : B \quad \mathsf{prod}(B) = A_1 \times A_2}{\Gamma \vdash_{\mathsf{S}} \mathsf{fst}\, t : A_1}\ \times_{e_1} \qquad \frac{\Gamma \vdash_{\mathsf{S}} t : B \quad \mathsf{prod}(B) = A_1 \times A_2}{\Gamma \vdash_{\mathsf{S}} \mathsf{snd}\, t : A_2}\ \times_{e_2}$$

$$\frac{\Gamma, x : A \vdash_{\mathsf{S}} t : B}{\Gamma \vdash_{\mathsf{S}} \lambda x : A_1.t : A \to B}\ \to \qquad \frac{\begin{array}{l}\Gamma \vdash_{\mathsf{S}} t_1 : C \\ \Gamma \vdash_{\mathsf{S}} t_2 : A_2 \quad A_2 \sim A_1 \quad \mathsf{fun}(C) = A_1 \to B_1\end{array}}{\Gamma \vdash_{\mathsf{S}} t_1\, t_2 : B_1}\ \to_e$$

Type Consistency:

$$\frac{}{A \sim A}\ \mathrm{refl} \qquad \frac{}{A \sim ?}\ \mathrm{box} \qquad \frac{}{? \sim A}\ \mathrm{unbox} \qquad \frac{A_1 \sim A_2 \quad B_1 \sim B_2}{A_1 \to B_1 \sim A_2 \to B_2}\ \to$$

$$\frac{A_1 \sim A_2 \quad B_1 \sim B_2}{A_1 \times B_1 \sim A_2 \times B_2}\ \times$$

Fig. 1. The gradually simply typed λ-calculus: $\lambda^{?}_{\to}$

As one can see these allow the unknown type to be split, otherwise they simply return the given product/arrow type. The typing rules also depend on type consistency which is a reflexive and symmetric binary relation. Intuitively, one can view these rules as indicating which types can be cast. That is, $A \sim B$ states that A can be cast into B and vice versa. As we will see in the interpretation type consistency amounts to requiring the existence of a casting morphism.

We can view gradual typing as a surface language feature much like type inference, and we give $\lambda^{?}_{\to}$ its operational semantics by translating it to a fully annotated core language – sometimes called the casting calculus – called $\lambda^{\Rightarrow}_{\to}$. The syntax, typing rules, and reduction relation for $\lambda^{\Rightarrow}_{\to}$ can be found in Figure 2. Its syntax is an extension of the syntax of $\lambda^{?}_{\to}$ where terms are the only syntactic class that differs, and so we do not repeat the syntax of types or contexts. The reduction relation is defined as a simple extension of call-by-value for the simply typed λ-calculus, and so we do not give all of the rules here, but rather only the casting rules. In addition, to save space we do not define the cast insertion algorithm for $\lambda^{?}_{\to}$. Please see Siek et al. [9] for its definition. To make it easier for the reader to connect the reduction rules to their interpretations we give the reduction rules for $\lambda^{\Rightarrow}_{\to}$ using the terms-in-context formulation; for an introduction to this style please see Crole [2].

The casting calculus $\lambda^{\Rightarrow}_{\to}$ also depends on type consistency. Now type consistency is not transitive, because in the surface language we do not want every type to

Syntax:

$$\begin{array}{llll} \text{(Atomic Types)} & T & ::= \mathsf{Unit} \mid \mathsf{Nat} & \text{(values)} \quad v ::= \lambda x : A.t \\ \text{(Ground Types)} & R & ::= T \mid ? \to ? & \text{(terms)} \quad t ::= \ldots \mid t : A \Rightarrow B \end{array}$$

Typing Rules:

$$\frac{x : A \in \Gamma}{\Gamma \vdash_{\mathsf{C}} x : A}\,\text{var} \qquad \frac{}{\Gamma \vdash_{\mathsf{C}} \mathsf{triv} : \mathsf{Unit}}\,\text{unit} \qquad \frac{}{\Gamma \vdash_{\mathsf{C}} 0 : \mathsf{Nat}}\,\text{zero} \qquad \frac{\Gamma \vdash_{\mathsf{C}} t : \mathsf{Nat}}{\Gamma \vdash_{\mathsf{C}} \mathsf{succ}\, t : \mathsf{Nat}}\,\text{succ}$$

$$\frac{\Gamma \vdash_{\mathsf{C}} t_1 : A_1 \quad \Gamma \vdash_{\mathsf{C}} t_2 : A_2}{\Gamma \vdash_{\mathsf{C}} (t_1, t_2) : A_1 \times A_2}\,\times \qquad \frac{\Gamma \vdash_{\mathsf{C}} t : A_1 \times A_2}{\Gamma \vdash_{\mathsf{C}} \mathsf{fst}\, t : A_1}\,\times_{e_1} \qquad \frac{\Gamma \vdash_{\mathsf{C}} t : A_1 \times A_2}{\Gamma \vdash_{\mathsf{C}} \mathsf{snd}\, t : A_2}\,\times_{e_2}$$

$$\frac{\Gamma, x : A \vdash_{\mathsf{C}} t : B}{\Gamma \vdash_{\mathsf{C}} \lambda x : A_1.t : A \to B}\,\to \qquad \frac{\Gamma \vdash_{\mathsf{C}} t_1 : A \to B \quad \Gamma \vdash_{\mathsf{C}} t_2 : A}{\Gamma \vdash_{\mathsf{C}} t_1\, t_2 : B}\,\to_e$$

$$\frac{\Gamma \vdash_{\mathsf{C}} t : A \quad A \sim B}{\Gamma \vdash_{\mathsf{C}} (t : A \Rightarrow B) : B}\,\text{cast}$$

Reduction Relation:

$$\frac{\Gamma \vdash_{\mathsf{C}} v : A}{\Gamma \vdash v \rightsquigarrow v : A}\,\text{values} \qquad \frac{\Gamma \vdash_{\mathsf{C}} v : T}{\Gamma \vdash v : T \Rightarrow T \rightsquigarrow v : T}\,\text{id-atom} \qquad \frac{\Gamma \vdash_{\mathsf{C}} v : ?}{\Gamma \vdash v : ? \Rightarrow ? \rightsquigarrow v : ?}\,\text{id-U}$$

$$\frac{\Gamma \vdash_{\mathsf{C}} v : R}{\Gamma \vdash v : R \Rightarrow ? \Rightarrow R \rightsquigarrow v : R}\,\text{succeed}$$

$$\frac{\Gamma \vdash_{\mathsf{C}} v_1 : A_1 \to B_1 \quad \Gamma \vdash_{\mathsf{C}} v_2 : A_2 \quad (A_1 \to B_1) \sim (A_2 \to B_2)}{\Gamma \vdash (v_1 : (A_1 \to B_1) \Rightarrow (A_2 \to B_2))\, v_2 \rightsquigarrow v_1\, (v_2 : A_2 \Rightarrow A_1) : B_1 \Rightarrow B_2 : B_2}\,\text{arrow}$$

$$\frac{\Gamma \vdash_{\mathsf{C}} v : A \quad A \sim R \quad A \neq R \quad A \neq ?}{\Gamma \vdash v : A \Rightarrow ? \rightsquigarrow v : A \Rightarrow R \Rightarrow ? : ?}\,\text{expand}_1$$

$$\frac{\Gamma \vdash_{\mathsf{C}} v : ? \quad A \sim R \quad A \neq R \quad A \neq ?}{\Gamma \vdash v : ? \Rightarrow A \rightsquigarrow v : ? \Rightarrow R \Rightarrow A : A}\,\text{expand}_2$$

Fig. 2. The core casting calculus: $\lambda^{\Rightarrow}_{\to}$

be consistent with every other type. However, the casting calculus allows one to compose consistency relations. For example, suppose $A \sim B$, $B \sim C$, and $\Gamma \vdash_{\mathsf{C}} t : A$, then using the rule cast of the casting calculus we may type $\Gamma \vdash_{\mathsf{C}} t : A \Rightarrow B \Rightarrow C : C$. This composition is the reason why we interpret type consistency as casting morphisms in the model.

3.1 The Interpretation

Next we show how to interpret $\lambda^{?}_{\to}$ and $\lambda^{\Rightarrow}_{\to}$ into a gradual λ-model. We will show how to interpret the typing of the former into the model, and then show how to do the same for the latter, furthermore, we show that reduction can be interpreted into the model as well, thus concluding soundness for $\lambda^{\Rightarrow}_{\to}$.

First, we must give the interpretation of types and contexts, but this interpretation is obvious, because we have been making sure to match the names of types and objects throughout this paper.

Definition 3.1 Suppose $(\mathcal{T}, \mathcal{C}, ?, \mathsf{T}, \mathsf{split}, \mathsf{squash}, \mathsf{box}, \mathsf{unbox})$ is a gradual λ-model. Then we define the interpretation of types into $\mathcal{C}$ as follows:

$$[\![?]\!] = ? \qquad [\![\mathsf{Unit}]\!] = 1 \qquad [\![A_1 \to A_2]\!] = [\![A_1]\!] \to [\![A_2]\!]$$
$$[\![\mathsf{Nat}]\!] = \mathsf{Nat} \qquad [\![A_1 \times A_2]\!] = [\![A_1]\!] \times [\![A_2]\!]$$

We extend this interpretation to typing contexts as follows:

$$[\![\cdot]\!] = 1 \qquad [\![\Gamma, x : A]\!] = [\![\Gamma]\!] \times [\![A]\!]$$

Throughout the remainder of this paper we will drop the interpretation symbols around types.

Before we can interpret the typing rules of $\lambda^{?}_{\to}$ and $\lambda^{\Rightarrow}_{\to}$ we must show how to interpret the consistency relation from Figure 1. These will correspond to casting morphisms.

Lemma 3.2 (Type Consistency in the Model) *Suppose* $(\mathcal{T}, \mathcal{C}, ?, \mathsf{T}, \mathsf{split}, \mathsf{squash}, \mathsf{box}, \mathsf{unbox})$ *is a gradual* λ*-model, and* $A \sim B$ *for some types* A *and* B*. Then there are two casting morphisms* $c_1 : A \longrightarrow B$ *and* $c_2 : B \longrightarrow A$*.*

Proof. This proof holds by induction on the form $A \sim B$ using the morphisms $\mathsf{Box}_A : A \longrightarrow ?$ and $\mathsf{Unbox}_A : ? \longrightarrow A$. Please see Appendix A.3 for the complete proof. □

Showing that both c_1 and c_2 exist corresponds to the fact that $A \sim B$ is symmetric.

At this point we have everything we need to show our main result which is that typing in both $\lambda^{?}_{\to}$ and $\lambda^{\Rightarrow}_{\to}$, and evaluation in $\lambda^{\Rightarrow}_{\to}$ can be interpreted into the categorical model.

Theorem 3.3 (Interpretation of Typing) *Suppose* $(\mathcal{T}, \mathcal{C}, ?, \mathsf{T}, \mathsf{split}, \mathsf{squash}, \mathsf{box}, \mathsf{unbox})$ *is a gradual* λ*-model. If* $\Gamma \vdash_{\mathsf{S}} t : A$ *or* $\Gamma \vdash_{\mathsf{C}} t : A$*, then there is a morphism* $[\![t]\!] : [\![\Gamma]\!] \longrightarrow A$ *in* $\mathcal{C}$*.*

Proof. Both parts of the proof hold by induction on the form of the assumed typing derivation, and uses most of the results we have developed up to this point. Please see Appendix A.4 for the complete proof. □

Theorem 3.4 (Interpretation of Evaluation) *Suppose* $(\mathcal{T}, \mathcal{C}, ?, \mathsf{T}, \mathsf{split}, \mathsf{squash}, \mathsf{box}, \mathsf{unbox})$ *is a gradual* λ*-model. If* $\Gamma \vdash t_1 \rightsquigarrow t_2 : A$*, then* $[\![t_1]\!] = [\![t_2]\!] : [\![\Gamma]\!] \longrightarrow A$*.*

Proof. This proof holds by induction on the form of $\Gamma \vdash t_1 \rightsquigarrow t_2 : A$, and uses Theorem 3.3, Lemma 3.2, and Corollary A.1. Please see Appendix A.5 for the complete proof. □

4 Simply Typed Grady

Just as the simply typed λ-calculus corresponds to cartesian closed categories our categorical model has a corresponding type theory we call Simply Typed Grady

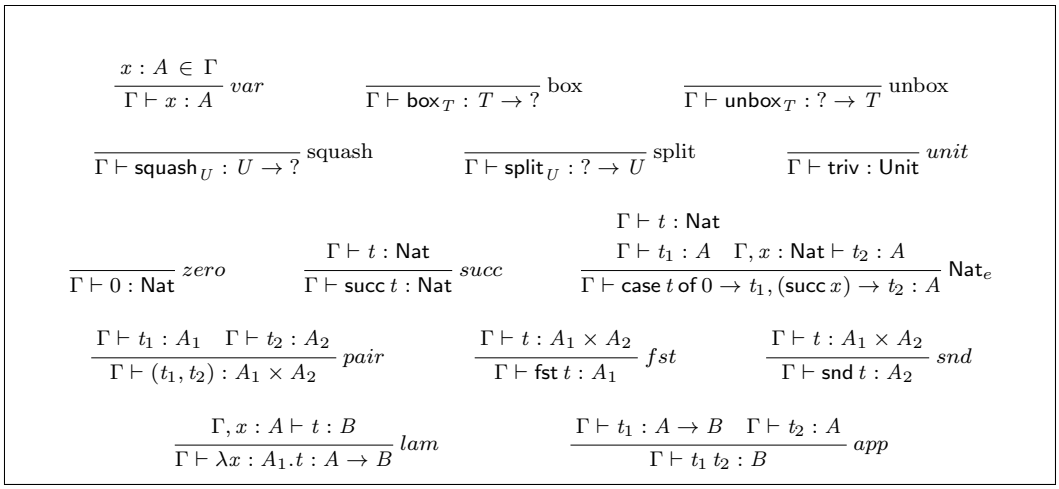


Fig. 3. Typing rules for Grady

(Grady). It consists of all of the structure found in the model. Its syntax is an extension of the syntax for $\lambda^{\Rightarrow}_{\to}$.

Definition 4.1 Syntax for Grady:

$$\begin{array}{rrcl}
\text{(basic skeletons)} & U & ::= & ? \to ? \mid ? \times ? \\
\text{(skeletons)} & S & ::= & ? \mid S_1 \times S_2 \mid S_1 \to S_2 \\
\text{(atomic types)} & C & ::= & \mathsf{Unit} \mid \mathsf{Nat} \\
\text{(terms)} & t & ::= & \ldots \mid \mathsf{split}_U \mid \mathsf{squash}_U \mid \mathsf{box}_T \mid \mathsf{unbox}_T \\
 & & & \mid \mathsf{case}\, t\, \mathsf{of}\, 0 \to t_1, (\mathsf{succ}\, x) \to t_2 \\
\text{(values)} & v & ::= & \ldots \mid \mathsf{squash}_U \mid \mathsf{box}_T
\end{array}$$

The typing rules for Grady can be found in Figure 3 and its reduction rules can be found in Figure 4. Reduction for Grady differs from $\lambda^{\Rightarrow}_{\to}$ in that it is an extended formulation of call-by-name. We only allow reduction under unbox and split, and we do not allow reduction under the branches of case.

Just as we did for the categorical model (Lemma 2.13) we can lift box_C and unbox_C to arbitrary type.

Lemma 4.2 (Syntactic Box_A and Unbox_A) *Given any type A there are functions Box_A and Unbox_A such that the following typing rules are admissible:*

$$\frac{}{\Gamma \vdash \mathsf{Box}_A : A \to ?}\ Box \qquad \frac{}{\Gamma \vdash \mathsf{Unbox}_A : ? \to A}\ Unbox$$

Furthermore, the following reduction rule is admissible:

$$\frac{\Gamma \vdash t : A}{\Gamma \vdash \mathsf{Unbox}_A\, (\mathsf{Box}_A\, t) \rightsquigarrow t : A}\ retract_3$$

$$\frac{x : A \in \Gamma}{\Gamma \vdash x \rightsquigarrow x : A}\ \text{var} \qquad \frac{\Gamma \vdash t : T}{\Gamma \vdash \mathsf{unbox}_T\,(\mathsf{box}_T\,t) \rightsquigarrow t : T}\ \text{retract}_1$$

$$\frac{\Gamma \vdash t_1 \rightsquigarrow t_2 : T}{\Gamma \vdash \mathsf{unbox}_T\,t_1 \rightsquigarrow \mathsf{unbox}_T\,t_2 : T}\ \text{unbox} \qquad \frac{\Gamma \vdash t : U}{\Gamma \vdash \mathsf{split}_U\,(\mathsf{squash}_U\,t) \rightsquigarrow t : U}\ \text{retract}_2$$

$$\frac{\Gamma \vdash t_1 \rightsquigarrow t_2 : U}{\Gamma \vdash \mathsf{split}_U\,t_1 \rightsquigarrow \mathsf{split}_U\,t_2 : U}\ \text{split} \qquad \frac{\Gamma, x : A_1 \vdash t_1 : A_2 \quad \Gamma \vdash t_2 : A_1}{\Gamma \vdash (\lambda x : A_1.t_1)\,t_2 \rightsquigarrow [t_2/x]t_2 : A_2}\ \beta$$

$$\frac{\Gamma \vdash t_1 : A_1 \quad \Gamma \vdash t_2 : A_2}{\Gamma \vdash \mathsf{fst}\,(t_1, t_2) \rightsquigarrow t_1 : A_1}\ \times_{e_1} \qquad \frac{\Gamma \vdash t_1 : A_1 \quad \Gamma \vdash t_2 : A_2}{\Gamma \vdash \mathsf{snd}\,(t_1, t_2) \rightsquigarrow t_2 : A_2}\ \times_{e_2} \qquad \frac{\Gamma \vdash t \rightsquigarrow t' : \mathsf{Nat}}{\Gamma \vdash \mathsf{succ}\,t \rightsquigarrow \mathsf{succ}\,t' : \mathsf{Nat}}\ \text{succ}$$

$$\frac{\Gamma \vdash t_1 : A \quad \Gamma, x : \mathsf{Nat} \vdash t_2 : A}{\Gamma \vdash \mathsf{case}\,0\,\mathsf{of}\,0 \to t_1, (\mathsf{succ}\,x) \to t_2 \rightsquigarrow t_1 : A}\ \mathsf{Nat}_{e_0}$$

$$\frac{\begin{array}{c}\Gamma \vdash t : \mathsf{Nat} \\ \Gamma \vdash t_1 : A \quad \Gamma, x : \mathsf{Nat} \vdash t_2 : A\end{array}}{\Gamma \vdash \mathsf{case}\,(\mathsf{succ}\,t)\,\mathsf{of}\,0 \to t_1, (\mathsf{succ}\,x) \to t_2 \rightsquigarrow [t/x]t_2 : A}\ \mathsf{Nat}_{e_1}$$

$$\frac{\begin{array}{c}\Gamma \vdash t \rightsquigarrow t' : \mathsf{Nat} \\ \Gamma \vdash t_1 : A \quad \Gamma, x : \mathsf{Nat} \vdash t_2 : A\end{array}}{\Gamma \vdash \mathsf{case}\,t\,\mathsf{of}\,0 \to t_1, (\mathsf{succ}\,x) \to t_2 \rightsquigarrow \mathsf{case}\,t'\,\mathsf{of}\,0 \to t_1, (\mathsf{succ}\,x) \to t_2 : A}\ \text{case}_1$$

$$\frac{\Gamma \vdash t_1 \rightsquigarrow t_1' : A_1 \to A_2 \quad \Gamma \vdash t_2 : A_1}{\Gamma \vdash t_1\,t_2 \rightsquigarrow t_1'\,t_2 : A_2}\ \to_{e_1} \qquad \frac{\Gamma \vdash v : A_1 \to A_2 \quad \Gamma \vdash t \rightsquigarrow t' : A_1}{\Gamma \vdash v\,t \rightsquigarrow v\,t' : A_2}\ \to_{e_2}$$

$$\frac{\Gamma \vdash t \rightsquigarrow t' : A_1 \times A_2}{\Gamma \vdash \mathsf{fst}\,t \rightsquigarrow \mathsf{fst}\,t' : A_1}\ \mathsf{fst} \qquad \frac{\Gamma \vdash t \rightsquigarrow t' : A_1 \times A_2}{\Gamma \vdash \mathsf{snd}\,t \rightsquigarrow \mathsf{snd}\,t' : A_2}\ \mathsf{snd} \qquad \frac{\Gamma \vdash t_1 \rightsquigarrow t_1' : A_1 \quad \Gamma \vdash t_2 : A_2}{\Gamma \vdash (t_1, t_2) \rightsquigarrow (t_1', t_2) : A_1 \times A_2}\ \times_1$$

$$\frac{\Gamma \vdash t_1 : A_1 \quad \Gamma \vdash t_2 \rightsquigarrow t_2' : A_2}{\Gamma \vdash (t_1, t_2) \rightsquigarrow (t_1, t_2') : A_1 \times A_2}\ \times_2$$

Fig. 4. Reduction rules for Grady

Proof. The functions Box_A and Unbox_A can be defined using the construction from the categorical model, e.g. Definition 2.7, Definition 2.10, and Lemma 2.13. However, the categorical notions of composition, identity, and the functors $- \to -$ and $- \times -$ must be defined as meta-functions first, but after they are, then the same constructions apply. Please see Appendix A.6 for the constructions. □

Similarly, we can lift split and squash to arbitrary skeletons.

Lemma 4.3 (Syntactic Split_S and Squash_S) *Given any type A there are functions Split_A and Squash_S such that the following typing rules are admissible:*

$$\frac{}{\Gamma \vdash \mathsf{Split}_S : S \to ?}\ \textit{Split} \qquad \frac{}{\Gamma \vdash \mathsf{Squash}_S : ? \to S}\ \textit{Squash}$$

Furthermore, the following reduction rule is admissible:

$$\frac{\Gamma \vdash t : S}{\Gamma \vdash \mathsf{Split}_S\,(\mathsf{Squash}_S\,t) \rightsquigarrow t : S}\ \textit{retract}_4$$

Perhaps unsurprisingly, due to our results with respect to the categorical model, we can use the previous results to construct a type-directed translation of both $\lambda^{?}_{\rightarrow}$ and $\lambda^{\Rightarrow}_{\rightarrow}$ into Grady.

Lemma 4.4 (Translations)

i. If $\Gamma \vdash t : A$ hold in either $\lambda^{?}_{\rightarrow}$ or $\lambda^{\Rightarrow}_{\rightarrow}$, then there exists a term t' such that $\Gamma \vdash t' : A$ holds in Grady.

ii. If $\Gamma \vdash t_1 \rightsquigarrow t_2 : A$ holds in $\lambda^{\Rightarrow}_{\rightarrow}$, then $\Gamma \vdash t'_1 \rightsquigarrow^* t'_2 : A$ holds in Grady, where $\Gamma \vdash t'_1 : A$ and $\Gamma \vdash t'_2 : A$ are both the corresponding Grady terms.

Proof. The proof of this result is similar to the proof that both $\lambda^{?}_{\rightarrow}$ and $\lambda^{\Rightarrow}_{\rightarrow}$ can be interpreted into the categorical model, Theorem 3.3 and Theorem 3.4, and thus, we do not give the full proof. The proof of part one holds by induction on $\Gamma \vdash t : A$, and using the realization that if $A \sim B$ for some types A and B then there are casting terms $\cdot \vdash c_1 : A \rightarrow B$ and $\cdot \vdash c_2 : B \rightarrow A$ following the proof of Lemma 3.2. The proof of part two holds by induction on $\Gamma \vdash t_1 \rightsquigarrow t_2 : A$ making use of part one; it is similar to the proof of Theorem 3.4. □

One important point of this system is it is straightforward to extend with new features, because it does not depend on type consistency. For example, in the sequel paper [6] we show that extending this system with bounded quantification is straightforward, but quite difficult for the gradually typed surface language. Furthermore, the different types of casts in Grady are tracked within the term which may lead to finer grain analysis of this system.

Just as Abadi et al. [1] argue it is quite useful to have access to the untyped λ-calculus. In the remainder of this section we give some example Grady programs utilizing this powerful feature. We have a full implementation of Grady with several extensions available [7]. All examples in this section can be typed and ran in the implementation, and thus, we make use of Grady's concrete syntax which is very similar to Haskell's and does not venture too far from the mathematical syntax given above.

Grady does not have a primitive notion of recursion, but it is well-known that we can define the Y combinator in the untyped λ-calculus. Its definition in Grady is as follows:

```
omega : (? → ?) → ?
omega = \(x : ? → ?) → (x (squash (? → ?) x));

fix : (? → ?) → ?
fix = \(f : ? → ?) → omega (\(x:?) → f ((split (? → ?) x) x));
```

Using fix we can define the usual arithmetic operations in Grady, but we use a typed version of fix.

```
fixNat : ((Nat → Nat) → (Nat → Nat)) → (Nat → Nat)
fixNat = \(f:(Nat → Nat) → (Nat → Nat)) →
         unbox⟨Nat → Nat⟩ (fix (\(y:?)→ box⟨Nat → Nat⟩ (f (unbox⟨Nat → Nat⟩ y))));

pred : Nat → Nat
```

[6] The working draft of the next paper on bounded quantification can be found at https://github.com/heades/gradual-typing/blob/MFPS17/Bounded/main.pdf.

[7] Please see http://metatheorem.org/gradual-typing/ for access to the implementation as well as full documentation on how to install and use it.

```
pred = \(n:Nat) → case n of 0 → 0, (succ n') → n';

add : Nat → Nat → Nat
add = \(m:Nat) → fixNat
     (\(r:Nat → Nat) →
      \(n:Nat) → case n of 0 → m, (succ n') → succ (r n'));

sub : Nat → Nat → Nat
sub = \(m:Nat) → fixNat
     (\(r:Nat → Nat) →
      \(n:Nat) → case n of 0 → m, (succ n') → pred (r n'));

mult : Nat → Nat → Nat
mult = \(m:Nat) → fixNat
     (\(r:Nat → Nat) →
      \(n:Nat) → case n of 0 → 0, (succ n') → add m (r n'));

exp : Nat → Nat → Nat
exp = \(m:Nat) → fixNat
     (\(r:Nat → Nat) →
      \(n:Nat) → case n of 0 → 1, (succ n') → mult m (r n'));
```

The body of each of the examples above are fully typed and the only dynamic typing that occurs is in the use of fix. Extending Grady with polymorphism would allow for the definition of fixNat to be abstracted, and then we could do statically typed recursion at any type. In fact, the implementation of Grady already supports this:

```
fixp : forall (X <: Simple).((X → X) → X)
fixp = \(X <: Simple) →
       \(f:X → X)    →
           unbox⟨X⟩ (fix (\(y:?)→ box⟨X⟩ (f (unbox⟨X⟩ y))));
```

The type bound Simple forces X to be a simple type. We do not allow casting of polymorphic types.

5 Conclusion

We have given a new categorical model that combines static and dynamic typing using the theory of retracts. Our model is an extension of Scott's [7] model of the untyped λ-calculus. Then we showed that Siek and Taha's gradually typed λ-calculus [9] can be soundly interpreted into our model. Finally, we define the corresponding typed λ-calculus called Grady that corresponds to our model through the Curry-Howard-Lambek correspondence.

Future work. Gradual typing reduces the number of explicit casts into the untyped fragment substantially for the programmer. However, one open question is how can gradual typing be extended with polymorphism? In a follow up paper we show how to extend gradual typing with bounded quantification. The bounds can be used to limit which types are castable to the unknown type. For example, we will not allow the programmer to cast a polymorphic type to the unknown type, because we do not have a good model for this, and we are not convinced it would be useful. The core language of the gradual type system is Grady. Finally, we show that this system satisfies the gradual guarantee as laid out by Siek et al. [9]. Adding bounded quantification is a non-trivial extension, and thus, proving the gradual guarantee was quite laborious. The proofs found in the literature for the gradual guarantee usually make heavy use of inversion for typing, but for a system with polymorphism and subtyping this can be difficult. Instead of using inversion for typing we show

that one can often use inversion for other judgments and make the proof effort more tractable.

References

[1] Abadi, M., L. Cardelli, B. Pierce and G. Plotkin, *Dynamic typing in a statically-typed language*, in: *Proceedings of the 16th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '89 (1989), pp. 213–227.

[2] Crole, R. L., "Categories for Types," Cambridge University Press, 1994.

[3] Garcia, R., A. M. Clark and E. Tanter, *Abstracting gradual typing*, in: *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '16 (2016), pp. 429–442.

[4] Henglein, F., *Dynamic typing: syntax and proof theory*, Science of Computer Programming **22** (1994), pp. 197 – 230.

[5] Lambek, J., *From lambda calculus to cartesian closed categories*, To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism (1980), pp. 376–402.

[6] Lambek, J. and P. Scott, "Introduction to Higher-Order Categorical Logic," Cambridge Studies in Advanced Mathematics, Cambridge University Press, 1988.

[7] Scott, D., *Relating theories of the lambda-calculus*, in: *To H.B. Curry: Essays on Combinatory Logic, Lambda-Calculus and Formalism (eds. Hindley and Seldin)* (1980), pp. 403–450.

[8] Siek, J. G. and W. Taha, *Gradual typing for functional languages*, in: *Scheme and Functional Programming Workshop*, 1 **6**, 2006, pp. 81–92.

[9] Siek, J. G., M. M. Vitousek, M. Cimini and J. T. Boyland, *Refined Criteria for Gradual Typing*, in: T. Ball, R. Bodik, S. Krishnamurthi, B. S. Lerner and G. Morrisett, editors, *1st Summit on Advances in Programming Languages (SNAPL 2015)*, Leibniz International Proceedings in Informatics (LIPIcs) **32** (2015), pp. 274–293.

A Proofs

A.1 Proof of Lifted Retract (Lemma 2.8)

This is a proof by induction on the form of A.

Case. Suppose A is atomic. Then:

$$\widehat{\mathsf{box}_A}; \widehat{\mathsf{unbox}_A} = \mathsf{box}_A; \mathsf{unbox}_A = \mathsf{id}_A$$

Case. Suppose A is ?. Then:

$$\begin{aligned} \widehat{\mathsf{box}_A}; \widehat{\mathsf{unbox}_A} &= \widehat{\mathsf{box}_?}; \widehat{\mathsf{unbox}_?} \\ &= \mathsf{id}_?; \mathsf{id}_? \\ &= \mathsf{id}_? \\ &= \mathsf{id}_A \end{aligned}$$

Case. Suppose $A = A_1 \to A_2$. Then:

$$\begin{aligned} \widehat{\mathsf{box}_A}; \widehat{\mathsf{unbox}_A} &= \widehat{\mathsf{box}_{(A_1 \to A_2)}}; \widehat{\mathsf{unbox}_{(A_1 \to A_2)}} \\ &= (\widehat{\mathsf{unbox}_{A_1}} \to \widehat{\mathsf{box}_{A_2}}); (\widehat{\mathsf{box}_{A_1}} \to \widehat{\mathsf{box}_{A_2}}) \\ &= (\widehat{\mathsf{box}_{A_1}}; \widehat{\mathsf{unbox}_{A_1}}) \to (\widehat{\mathsf{box}_{A_2}}; \widehat{\mathsf{unbox}_{A_2}}) \end{aligned}$$

By two applications of the induction hypothesis we know the following:

$$\widehat{\mathsf{box}}_{A_1}; \widehat{\mathsf{unbox}}_{A_1} = \mathsf{id}_{A_1} \quad \text{and} \quad \widehat{\mathsf{box}}_{A_2}; \widehat{\mathsf{unbox}}_{A_2} = \mathsf{id}_{A_2}$$

Thus, we know the following:

$$\begin{aligned}(\widehat{\mathsf{box}}_{A_1}; \widehat{\mathsf{unbox}}_{A_1}) \to (\widehat{\mathsf{box}}_{A_2}; \widehat{\mathsf{unbox}}_{A_2}) &= \mathsf{id}_{A_1} \to \mathsf{id}_{A_2} \\ &= \mathsf{id}_{A_1 \to A_2} \\ &= \mathsf{id}_A\end{aligned}$$

Case. Suppose $A = A_1 \times A_2$. Then:

$$\begin{aligned}\widehat{\mathsf{box}}_A; \widehat{\mathsf{unbox}}_A &= \widehat{\mathsf{box}}_{(A_1 \times A_2)}; \widehat{\mathsf{unbox}}_{(A_1 \times A_2)} \\ &= (\widehat{\mathsf{box}}_{A_1} \times \widehat{\mathsf{box}}_{A_2}); (\widehat{\mathsf{unbox}}_{A_1} \times \widehat{\mathsf{box}}_{A_2}) \\ &= (\widehat{\mathsf{box}}_{A_1}; \widehat{\mathsf{unbox}}_{A_1}) \times (\widehat{\mathsf{box}}_{A_2}; \widehat{\mathsf{unbox}}_{A_2})\end{aligned}$$

By two applications of the induction hypothesis we know the following:

$$\widehat{\mathsf{box}}_{A_1}; \widehat{\mathsf{unbox}}_{A_1} = \mathsf{id}_{A_1} \quad \text{and} \quad \widehat{\mathsf{box}}_{A_2}; \widehat{\mathsf{unbox}}_{A_2} = \mathsf{id}_{A_2}$$

Thus, we know the following:

$$\begin{aligned}(\widehat{\mathsf{box}}_{A_1}; \widehat{\mathsf{unbox}}_{A_1}) \times (\widehat{\mathsf{box}}_{A_2}; \widehat{\mathsf{unbox}}_{A_2}) &= \mathsf{id}_{A_1} \times \mathsf{id}_{A_2} \\ &= \mathsf{id}_{A_1 \times A_2} \\ &= \mathsf{id}_A\end{aligned}$$

A.2 Proof of Lemma 2.9

We must show that the function

$$\mathsf{S}_{A,B} : \mathsf{Hom}_{\mathcal{C}}(A, B) \longrightarrow \mathsf{Hom}_{\mathcal{S}}(\mathsf{S}A, \mathsf{S}B)$$

is injective.

So suppose $f \in \mathsf{Hom}_{\mathcal{C}}(A, B)$ and $g \in \mathsf{Hom}_{\mathcal{C}}(A, B)$ such that $\mathsf{S}f = \mathsf{S}g : \mathsf{S}A \longrightarrow \mathsf{S}B$. Then we can easily see that:

$$\begin{aligned}\mathsf{S}f &= \widehat{\mathsf{unbox}}_A; f; \widehat{\mathsf{box}}_B \\ &= \widehat{\mathsf{unbox}}_A; g; \widehat{\mathsf{box}}_B \\ &= \mathsf{S}g\end{aligned}$$

But, we have the following equalities:

$$\begin{aligned}
\widehat{\mathsf{unbox}}_A; f; \widehat{\mathsf{box}}_B &= \widehat{\mathsf{unbox}}_A; g; \widehat{\mathsf{box}}_B \\
\widehat{\mathsf{box}}_A; \widehat{\mathsf{unbox}}_A; f; \widehat{\mathsf{box}}_B; \widehat{\mathsf{unbox}}_B &= \widehat{\mathsf{box}}_A; \widehat{\mathsf{unbox}}_A; g; \widehat{\mathsf{box}}_B; \widehat{\mathsf{unbox}}_B \\
\mathsf{id}_A; f; \widehat{\mathsf{box}}_B; \widehat{\mathsf{unbox}}_B &= \mathsf{id}_A; g; \widehat{\mathsf{box}}_B; \widehat{\mathsf{unbox}}_B \\
\mathsf{id}_A; f; \mathsf{id}_B &= \mathsf{id}_A; g; \mathsf{id}_B \\
f &= g
\end{aligned}$$

The previous equalities hold due to Lemma 2.8.

A.3 Proof of Type Consistency in the Model (Lemma 3.2)

This is a proof by induction on the form of $A \sim B$.

Case.

$$\frac{}{A \sim A}\ \text{refl}$$

Choose $c_1 = c_2 = \mathsf{id}_A : A \longrightarrow A$.

Case.

$$\frac{}{A \sim ?}\ \text{box}$$

Choose $c_1 = \mathsf{Box}_A : A \longrightarrow ?$ and $c_2 = \mathsf{Unbox}_A : ? \to A$.

Case.

$$\frac{}{? \sim A}\ \text{unbox}$$

Choose $c_1 = \mathsf{Unbox}_A : ? \longrightarrow A$ and $c_2 = \mathsf{Box}_A : A \to ?$.

Case.

$$\frac{A_1 \sim A_2 \quad B_1 \sim B_2}{A_1 \to B_1 \sim A_2 \to B_2}\ \to$$

By the induction hypothesis there exists four casting morphisms $c_1' : A_1 \longrightarrow A_2$, $c_2' : A_2 \longrightarrow A_1$, $c_3' : B_1 \longrightarrow B_2$, and $c_4' : B_2 \longrightarrow B_1$. Choose $c_1 = c_2' \to c_3' : (A_1 \to B_1) \longrightarrow (A_2 \to B_2)$ and $c_2 = c_1' \to c_4' : (A_2 \to B_2) \longrightarrow (A_1 \to B_1)$.

Case.

$$\frac{A_1 \sim A_2 \quad B_1 \sim B_2}{A_1 \times B_1 \sim A_2 \times B_2}\ \times$$

By the induction hypothesis there exists four casting morphisms $c_1' : A_1 \longrightarrow A_2$, $c_2' : A_2 \longrightarrow A_1$, $c_3' : B_1 \longrightarrow B_2$, and $c_4' : B_2 \longrightarrow B_1$. Choose $c_1 = c_1' \times c_3' : A_1 \times B_1 \longrightarrow A_2 \times B_2$ and $c_2 = c_2' \times c_4' : A_2 \times B_2 \longrightarrow A_1 \times B_1$.

A.4 Proof of Interpretation of Types Theorem 3.3

First, we show how to interpret the rules of $\lambda^{?}_{\to}$, and then $\lambda^{\rightrightarrows}_{\to}$. This is a proof by induction on $\Gamma \vdash_{\mathsf{S}} t : A$.

Case.

$$\frac{x : A \in \Gamma}{\Gamma \vdash_{\mathsf{S}} x : A}\ \text{var}$$

Suppose with out loss of generality that $[\![\Gamma]\!] = A_1 \times \cdots \times A_i \times \cdots \times A_j$ where $A_i = A$. We know that $j > 0$ or the assumed typing derivation would not hold. Then take $[\![x]\!] = \pi_i : [\![\Gamma]\!] \longrightarrow A$.

Case.

$$\frac{}{\Gamma \vdash_{\mathsf{S}} \mathsf{triv} : \mathsf{Unit}}\ \text{unit}$$

Take $[\![\mathsf{triv}]\!] = \diamond_{[\![\Gamma]\!]} : [\![\Gamma]\!] \longrightarrow 1$ where $\diamond_{[\![\Gamma]\!]}$ is the unique terminal arrow that exists because $\mathcal{C}$ is cartesian closed.

Case.

$$\frac{}{\Gamma \vdash_{\mathsf{S}} 0 : \mathsf{Nat}}\ \text{zero}$$

Take $[\![0]\!] = \diamond_{[\![\Gamma]\!]}; \mathsf{z} : [\![\Gamma]\!] \longrightarrow \mathsf{Nat}$ where $\mathsf{z} : 1 \longrightarrow \mathsf{Nat}$ exists because $\mathcal{C}$ has a SNNO.

Case.

$$\frac{\Gamma \vdash_{\mathsf{S}} t : A \quad \mathsf{nat}(A) = \mathsf{Nat}}{\Gamma \vdash_{\mathsf{S}} \mathsf{succ}\ t : \mathsf{Nat}}\ \text{succ}$$

First, by the induction hypothesis there is a morphism $[\![t]\!] : [\![\Gamma]\!] \longrightarrow A$. Now we have two cases to consider, one when $A = ?$ and one when $A = \mathsf{Nat}$. Consider the former. Then interpret $[\![\mathsf{succ}\ t]\!] = [\![t]\!]; \mathsf{unbox}_{\mathsf{Nat}}; \mathsf{succ} : [\![\Gamma]\!] \longrightarrow \mathsf{Nat}$ where $\mathsf{succ} : \mathsf{Nat} \longrightarrow \mathsf{Nat}$ exists because $\mathcal{C}$ has a SNNO. Similarly, when $A = \mathsf{Nat}$, $[\![\mathsf{succ}\ t]\!] = [\![t]\!]; \mathsf{succ} : [\![\Gamma]\!] \longrightarrow \mathsf{Nat}$.

Case.

$$\frac{\Gamma \vdash_{\mathsf{S}} t_1 : A_1 \quad \Gamma \vdash_{\mathsf{S}} t_2 : A_2}{\Gamma \vdash_{\mathsf{S}} (t_1, t_2) : A_1 \times A_2}\ \times$$

By two applications of the induction hypothesis there are two morphisms $[\![t_1]\!] : [\![\Gamma]\!] \longrightarrow A$ and $[\![t_2]\!] : [\![\Gamma]\!] \longrightarrow B$. Then using the fact that $\mathcal{C}$ is cartesian we take $[\![(t_1, t_2)]\!] = \langle [\![t_1]\!], [\![t_2]\!] \rangle : [\![\Gamma]\!] \longrightarrow A \times B$.

Case.

$$\frac{\Gamma \vdash_{\mathsf{S}} t : B \quad \mathsf{prod}(B) = A_1 \times A_2}{\Gamma \vdash_{\mathsf{S}} \mathsf{fst}\ t : A_1}\ \times_{e_1}$$

First, by the induction hypothesis there is a morphism $[\![t]\!] : [\![\Gamma]\!] \longrightarrow B$. Now we have two cases to consider, one when $B = ?$ and one when $B = A_1 \times A_2$ for some types A_1 and A_2. Consider the former. We then know that it must be the case that $A_1 \times A_2 = ? \times ?$. Thus, we obtain the following interpretation $[\![\mathsf{fst}\ t]\!] = [\![t]\!]; \mathsf{split}_{(? \times ?)}; \pi_1 : [\![\Gamma]\!] \longrightarrow ?$. Similarly, when $B = A_1 \times A_2$, then $[\![\mathsf{fst}\ t]\!] = [\![t]\!]; \pi_1 : [\![\Gamma]\!] \longrightarrow A_1$.

Case.

$$\frac{\Gamma \vdash_{\mathsf{S}} t : B \quad \mathsf{prod}(B) = A_1 \times A_2}{\Gamma \vdash_{\mathsf{S}} \mathsf{snd}\ t : A_2}\ \times_{e_2}$$

First, by the induction hypothesis there is a morphism $[\![t]\!] : [\![\Gamma]\!] \longrightarrow B$. Now we have two cases to consider, one when $B = ?$ and one when $B = A_1 \times A_2$ for some types A_1 and A_2. Consider the former. We then know that it must be the case that $A_1 \times A_2 = ? \times ?$. Thus, we obtain the following interpretation

$[\![\mathsf{snd}\ t]\!] = [\![t]\!]; \mathsf{split}_{(?\times ?)}; \pi_2 : [\![\Gamma]\!] \longrightarrow ?$. Similarly, when $B = A_1 \times A_2$, then $[\![\mathsf{snd}\ t]\!] = [\![t]\!]; \pi_2 : [\![\Gamma]\!] \longrightarrow A_2$.

Case.

$$\frac{\Gamma, x : A \vdash_{\mathsf{S}} t : B}{\Gamma \vdash_{\mathsf{S}} \lambda x : A_1.t : A \to B} \to$$

By the induction hypothesis there is a morphism $[\![t]\!] : [\![\Gamma]\!] \times A \longrightarrow B$. Then take $[\![\lambda x : A.t]\!] = \mathsf{curry}([\![t]\!]) : [\![\Gamma]\!] \longrightarrow (A \to B)$, where $\mathsf{curry} : \mathsf{Hom}_{\mathcal{C}}(X \times Y, Z) \longrightarrow \mathsf{Hom}_{\mathcal{C}}(X, Y \to Z)$ exists because $\mathcal{C}$ is closed.

Case.

$$\frac{\Gamma \vdash_{\mathsf{S}} t_1 : C \qquad \Gamma \vdash_{\mathsf{S}} t_2 : A_2 \quad A_2 \sim A_1 \quad \mathsf{fun}(C) = A_1 \to B_1}{\Gamma \vdash_{\mathsf{S}} t_1\ t_2 : B_1} \to_e$$

By the induction hypothesis there are two morphisms $[\![t_1]\!] : [\![\Gamma]\!] \longrightarrow C$ and $[\![t_2]\!] : [\![\Gamma]\!] \longrightarrow A_2$. In addition, by assumption we know that $A_2 \sim A_1$, and hence, by type consistency in the model (Lemma 3.2) there are casting morphisms $c_1 : A_2 \longrightarrow A_1$ and $c_2 : A_1 \longrightarrow A_2$. We have two cases to consider, one when $C = ?$ and one when $C = A_1 \to B_1$. Consider the former. Then we have the interpretation

$$[\![t_1\ t_2]\!] = \langle [\![t_1]\!]; \mathsf{split}_{(?\to ?)}, [\![t_2]\!]; c_1 \rangle; \mathsf{app}_{A_2,B_1} : [\![\Gamma]\!] \longrightarrow B_1.$$

Similarly, for the case when $C = A_1 \to B_1$ we have the interpretation

$$[\![t_1\ t_2]\!] = \langle [\![t_1]\!], [\![t_2]\!]; c_1 \rangle; \mathsf{app}_{A_1,B_1} : [\![\Gamma]\!] \longrightarrow B_1.$$

Note that $\mathsf{app}_{A_1,B_1} : (A_1 \to B_1) \times A_1 \longrightarrow B_1$ exists because the model is cartesian closed.

Next we turn to $\lambda^{\Rightarrow}_{\to}$, but we do not show every rule, because it corresponds to the simply typed λ-calculus whose interpretation is similar to what we have already shown above except without casting morphism, and so we only show the case for the cast rule.

Case.

$$\frac{\Gamma \vdash_{\mathsf{C}} t : A \quad A \sim B}{\Gamma \vdash_{\mathsf{C}} (t : A \Rightarrow B) : B}\ \mathrm{cast}$$

By the induction hypothesis there is a morphism $[\![t]\!] : [\![\Gamma]\!] \longrightarrow A$, and by type consistency in the model (Lemma 3.2) there is a casting morphism $c_1 : A \longrightarrow B$. So take $[\![t : A \Rightarrow B]\!] = [\![t]\!]; c_1 : [\![\Gamma]\!] \longrightarrow B$.

A.5 Proof of Interpretation of Evaluation (Theorem 3.4)

This proof requires the following corollary to Lemma 3.2.

Corollary A.1 *Suppose* $(\mathcal{T}, \mathcal{C}, ?, \mathsf{T}, \mathsf{split}, \mathsf{squash}, \mathsf{box}, \mathsf{unbox})$ *is a gradual* λ*-model. Then we know the following:*

i. If $A \sim A$*, then* $c_1 = c_2 = \mathsf{id}_A : A \longrightarrow A$.

ii. *If $A \sim ?$, then there are casting morphisms:*

$$c_1 = \mathsf{Box}_A : A \longrightarrow ? \qquad c_2 = \mathsf{Unbox}_A : ? \longrightarrow A$$

iii. *If $? \sim A$, then there are casting morphisms:*

$$c_1 = \mathsf{Unbox}_A : ? \longrightarrow A \qquad c_2 = \mathsf{Box}_A : A \longrightarrow ?$$

iv. *If $A_1 \to B_1 \sim A_2 \to B_2$, then there are casting morphisms:*

$$\begin{aligned} c &= c_1 \to c_2 : (A_1 \to B_1) \longrightarrow (A_2 \to B_2) \\ c' &= c_3 \to c_4 : (A_2 \to B_2) \longrightarrow (A_1 \to B_1) \end{aligned}$$

where $c_1 : A_2 \longrightarrow A_1$ and $c_2 : B_1 \longrightarrow B_2$, and $c_3 : A_1 \longrightarrow A_2$ and $c_4 : B_2 \longrightarrow B_1$.

v. *If $A_1 \times B_1 \sim A_2 \times B_2$, then there are casting morphisms:*

$$\begin{aligned} c &= c_1 \times c_2 : (A_1 \times B_1) \longrightarrow (A_2 \times B_2) \\ c' &= c_3 \times c_4 : (A_2 \times B_2) \longrightarrow (A_1 \times B_1) \end{aligned}$$

where $c_1 : A_1 \longrightarrow A_2$ and $c_2 : B_1 \longrightarrow B_2$, and $c_3 : A_2 \longrightarrow A_1$ and $c_4 : B_2 \longrightarrow B_1$.

Proof. This proof holds by the construction of the casting morphisms from the proof of the previous result, and the fact that the type consistency rules are unique for each type. □

This proof holds by induction on the form of $\Gamma \vdash t_1 \rightsquigarrow t_2 : A$. We only show the cases for the casting rules, because the others are well-known to hold within any cartesian closed category; see [5] or [2]. We will routinely use Theorem 3.3 throughout this proof without mention.

Case.

$$\frac{\Gamma \vdash_{\mathsf{C}} v : T}{\Gamma \vdash v : T \Rightarrow T \rightsquigarrow v : T}\ \text{ID-ATOM}$$

We have a morphism $[\![v : T \Rightarrow T]\!] = [\![v]\!]; c_1 : [\![\Gamma]\!] \longrightarrow T$ based on the interpretation of typing where $c_1 : T \longrightarrow T$, but it must be the case that $c_1 = \mathsf{id}_T : T \longrightarrow T$ by Corollary A.1. Thus, $[\![v : T \Rightarrow T]\!] = [\![v]\!]; c_1 = [\![v]\!] : [\![\Gamma]\!] \longrightarrow T$.

Case.

$$\frac{\Gamma \vdash_{\mathsf{C}} v : ?}{\Gamma \vdash v : ? \Rightarrow ? \rightsquigarrow v : ?}\ \text{ID-U}$$

Similar to the previous case.

Case.

$$\frac{\Gamma \vdash_{\mathsf{C}} v : R}{\Gamma \vdash v : R \Rightarrow ? \Rightarrow R \rightsquigarrow v : R}\ \textsc{succeed}$$

The typing derivation for $v : R \Rightarrow ? \Rightarrow R$ is as follows:

$$\frac{\dfrac{\Gamma \vdash_{\mathsf{C}} v : R \quad R \sim ?}{\Gamma \vdash_{\mathsf{C}} v : R \Rightarrow ? : ?} \qquad ? \sim R}{\Gamma \vdash_{\mathsf{C}} v : R \Rightarrow ? \Rightarrow R : R}$$

By the induction hypothesis we have the morphism $[\![v]\!] : [\![\Gamma]\!] \longrightarrow R$. As we can see we will first use Box_R and then Unbox_R based on Corollary A.1. Thus, $[\![v : R \Rightarrow ? \Rightarrow R]\!] = [\![v]\!]; \mathsf{Box}_R; \mathsf{Unbox}_R = [\![v]\!]$, because Box_R and Unbox_R form a retract.

Case.

$$\frac{\Gamma \vdash_{\mathsf{C}} v_1 : A_1 \to B_1 \quad \Gamma \vdash_{\mathsf{C}} v_2 : A_2 \quad (A_1 \to B_1) \sim (A_2 \to B_2)}{\Gamma \vdash (v_1 : (A_1 \to B_1) \Rightarrow (A_2 \to B_2))\, v_2 \rightsquigarrow v_1\, (v_2 : A_2 \Rightarrow A_1) : B_1 \Rightarrow B_2 : B_2}\ \textsc{arrow}$$

By the induction hypothesis and Corollary A.1 we have the following morphisms:

$$[\![v_1]\!] : [\![\Gamma]\!] \longrightarrow (A_1 \to B_1)$$
$$[\![v_2]\!] : [\![\Gamma]\!] \longrightarrow A_2$$
$$c_1 : A_2 \longrightarrow A_1$$
$$c_2 : B_1 \longrightarrow B_2$$

We must show the following:

$$\begin{aligned}[\![(v_1 : (A_1 \to B_1) \Rightarrow (A_2 \to B_2))\, v_2]\!] &= \langle [\![v_1]\!]; (c_1 \to c_2), [\![v_2]\!] \rangle; \mathsf{app}_{A_2,B_2} \\ &= \langle [\![v_1]\!], [\![v_2]\!]; c_1 \rangle; \mathsf{app}_{A_1,B_1}; c_2\end{aligned}$$

The previous equation holds as follows:

$$\begin{aligned}
&\langle [\![v_1]\!], [\![v_2]\!]; c_1 \rangle; \mathsf{app}_{A_1,B_1}; c_2 && \\
&= \langle [\![v_1]\!], [\![v_2]\!] \rangle; (\mathsf{id}_{A_1 \to B_1} \times c_1); \mathsf{app}_{A_1,B_1}; c_2 && \text{(Cartesian Products)} \\
&= \langle [\![v_1]\!], [\![v_2]\!] \rangle; (\mathsf{id}_{A_1 \to B_1} \times c_1); ((\mathsf{id}_{A_1} \to c_2) \times \mathsf{id}_{A_1}); \mathsf{app}_{A_1,B_2} && \text{(Naturality)} \\
&= \langle [\![v_1]\!], [\![v_2]\!] \rangle; (\mathsf{curry}((\mathsf{id}_{A_1 \to B_1} \times c_1); ((\mathsf{id}_{A_1} \to c_2) \times \mathsf{id}_{A_1}); \mathsf{app}_{A_1,B_2}) \times \mathsf{id}_{A_2}); \mathsf{app}_{A_2,B_2} && \text{(Closure)} \\
&= \langle [\![v_1]\!], [\![v_2]\!] \rangle; ((c_1 \to c_2) \times \mathsf{id}_{A_2}); \mathsf{app}_{A_2,B_2} && \text{(Closure)} \\
&= \langle [\![v_1]\!]; (c_1 \to c_2), [\![v_2]\!]; \mathsf{id}_{A_2} \rangle; \mathsf{app}_{A_2,B_2} && \text{(Cartesian)} \\
&= \langle [\![v_1]\!]; (c_1 \to c_2), [\![v_2]\!] \rangle; \mathsf{app}_{A_2,B_2} &&
\end{aligned}$$

Case.

$$\frac{\Gamma \vdash_{\mathsf{C}} v : A \quad A \sim R \quad A \neq R \quad A \neq ?}{\Gamma \vdash v : A \Rightarrow ? \rightsquigarrow v : A \Rightarrow R \Rightarrow ? : ?}\ \textsc{expand}_1$$

By the induction hypothesis and Lemma 3.2 we have the following morphisms:

$$[\![v]\!] : [\![\Gamma]\!] \longrightarrow A$$
$$c_1 : A \longrightarrow R$$

We must show the following:

$$\begin{aligned}[\![v : A \Rightarrow ?]\!] &= [\![v]\!]; \mathsf{Box}_A \\ &= [\![v]\!]; c_1; \mathsf{Box}_R\end{aligned}$$

However, notice that given the constraints above, it must be the case that $R = T$ or $R = ? \to ?$. If the former is true, then $A = T$ by the definition of consistency and the constraints above, but this implies that $c_1 = \mathsf{id}_T$ by Corollary A.1, and the result follows. However, consider the case when $R = ? \to ?$. Then given the constraints above $A = A_1 \to A_2$. Thus, $c_1 = (\mathsf{unbox}_{A_1} \to \mathsf{box}_{A_2})$ by Corollary A.1. In addition, it must be the case that $\mathsf{Box}_R = \mathsf{squash}_{(?\to?)}$ by the definition of Box_R, but by inspection of the definition of Box_A we have the following:

$$\begin{aligned}\mathsf{Box}_A &= \mathsf{Box}_{(A_1 \to A_2)} \\ &= (\mathsf{unbox}_{A_1} \to \mathsf{box}_{A_2}); \mathsf{squash}_{(?\to?)} \\ &= c_1; \mathsf{squash}_{(?\to?)} \\ &= c_1; \mathsf{Box}_R\end{aligned}$$

Thus, we obtain our result.

Case.

$$\frac{\Gamma \vdash_{\mathsf{C}} v : ? \quad A \sim R \quad A \neq R \quad A \neq ?}{\Gamma \vdash v : ? \Rightarrow A \rightsquigarrow v : ? \Rightarrow R \Rightarrow A : A}\ \textsc{expand}_2$$

This case is similar to the previous case, except that the interpretation uses Unbox_A and Unbox_R instead of Box_A and Box_R.

A.6 Proof of Lemma 4.3

First, we define the identify meta-function:

$$\mathsf{id}_A := \lambda x : A.x$$

Then composition. Suppose $\Gamma \vdash t_1 : A \to B$ and $\Gamma \vdash t_2 : B \to D$ are two terms, then we define:

$$t_1; t_2 := \lambda x : A.t_2\,(t_1\,x)$$

It is easy to see that the following rule is admissible:

$$\frac{\Gamma \vdash t_1 : A \to B \qquad \Gamma \vdash t_2 : B \to D}{\Gamma \vdash t_1; t_2 : A \to D}\ \textsc{comp}$$

The functor $- \times -$ requires two morphisms $\Gamma \vdash t_1 : A \to D$ and $\Gamma \vdash t_2 : B \to E$, and is defined as follows:

$$t_1 \times t_2 := \lambda x : A \times B.(t_1\,(\mathsf{fst}\,x), t_2\,(\mathsf{snd}\,x))$$

The following rule is admissible:

$$\frac{\Gamma \vdash t_1 : A \to D \qquad \Gamma \vdash t_2 : B \to E}{\Gamma \vdash t_1 \times t_2 : A \times B \to D \times E}\ \textsc{prod}$$

The functor $- \to -$ requires two morphisms $\Gamma \vdash t_1 : D \to A$ and $\Gamma \vdash t_2 : B \to E$, and is defined as follows:

$$t_1 \to t_2 := \lambda f : A \to B.\lambda y : D.t_2\,(f\,(t_1\,y))$$

The following rule is admissible:

$$\frac{\Gamma \vdash t_1 : D \to A \qquad \Gamma \vdash t_2 : B \to E}{\Gamma \vdash t_1 \to t_2 : (A \to B) \to (D \to E)}\ \textsc{prod}$$

At this point it is straightforward to carry out the definition of Box_A and Unbox_A using the definitions from the model. Showing the admissibility of the typing and reduction rules follows by induction on A.