1 Introduction, or, Ends: A Means to What End?

Consider a strongly-typed, functional programming language. Traditionally, we embed the semantics of such things into category theory by constructing a category where the objects are types of the language and morphisms are constructible functions: the composition closure of primitives and expressible \( \lambda \) terms. The category must be Cartesian-closed, if we want to embed functions, because we need exponential objects: they correspond to \( \to \)-typed objects.

In fact, there’s a natural tool that goes a great distance in helping us here. The \( \text{Hom}_C \) diagonal bifunctor. Given any two objects in \( C \), it gives us the set of arrows between them.

A glaring deficiency of this embedding, however, is the complete lack of polymorphism. There is no one “identity function” in this story, rather, there is one at every type. We can work around this by viewing prenex polymorphism (that is, all universal quantifiers scope over the entire type) as a metatheoretic instrument. We can adjust our proof rules which map a program to its categorical form to, essentially, defer computation of which categorical object or morphism is in play until after enough type information is available. Such treatments typically render the polymorphic identity function as \( (\lambda_{\alpha*} \lambda x. x) : \forall_{\alpha*} \alpha \to \alpha \), for example, making explicit the type-level binder. Hand waving a bit, the (usually STLC) program whose terms are terms of the language, \( \Lambda \)-bound functions, and (type) applications is then evaluated to obtain the underlying monomorphic program, which can then be given categorical semantics.

But that clever trick breaks down if we have non-prenex polymorphism, where we can actually give a binder whose type involves a universal quantifier. Such an expression might be \( (\Lambda_{\alpha*} \lambda f. \lambda x. f \alpha x) : \forall_{\alpha*} (\forall_{\beta*} \beta \to \beta) \to \alpha \to \alpha \). (This is not intended to be a terribly useful example, just illustrative.) Here we see that we have a type variable \( \alpha \) quantified in prenex position, and another, \( \beta \), whose quantifier cannot be moved higher. Put a different way, \( f \) is drawn from the set of polymorphic functions \( \forall_{\alpha*} \alpha \to \alpha \). If we want to give categorical semantics to this object, we need an object that represents that set.

If we think about it, though, we see that a particular term \( \Lambda_{\alpha*} \phi(\alpha) : \forall_{\alpha*} \tau(\alpha) \) is really a \( \alpha \)-indexed product (For each \( \alpha \in \alpha \), there is a \( \phi(\alpha) \).) Of course, there may be multiple \( \phi(\alpha) \) terms that all have the type \( \tau(\alpha) \), so the collection of all terms of type \( \forall_{\alpha*} \tau(\alpha) \) is a two-indexed collection of terms. This is the end!

2 Natural Transformations As Parametric Polymorphism

Any parametrically-polymorphic function, let’s call it \( \eta \), will have a type with a \( \forall_{\tau*} \) quantifier scoping over a \( \to \) constructor whose the two arguments are type-level functions of \( \tau \). That is, it will have type \( \forall_{\tau*} F \tau \to G \tau \), for some type-level functions \( F \) and \( G \).

In order for us to justify the term “parametric”, it should be the case that two instantiations of such a function, say, at types \( \tau \) and \( \tau' \), “behave similarly”. We can give an exact meaning to this when \( F \) and \( G \) are in fact functors, not just type-level functions: if there is some function \( F : \tau \to \tau' \), then these two paths of getting from \( F \tau \) to \( G \tau' \) must be equal if we are to say that \( \eta \tau \) and \( \eta \tau' \) “behave similarly”; this is just the definition of a natural transformation \( \eta : F \to G \):

The diagram is essentially unchanged if \( F \) and \( G \) are contravariant functors instead. I do not know how to formalize “parametric” when \( F \) and \( G \) are not functors (of either variance).

3 Profunctors

A profunctor is a bifunctor (it may be helpful to review \texttt{bifunctors.pdf}) with one argument of each type (typically, and here, the first is the contravariant one). That is, a profunctor \( S : A^{\text{op}} \times B \to C \) has two maps: \( S A B \), mapping pairs of objects to objects; and \( S f g : S A' B' \to S A B' \) mapping pairs of arrows to arrows, where \( A, A' \in A_0, B, B' \in B_0 \), \( f : A \to A' \in A_1 \), and \( g : B \to B' \in B_1 \). We know that the arrow map preserves identity and composition, though we will not actually need these facts.

\( ^{1} \) I believe this intuition describes an impredicative system, like System F, since \( * \) includes the object representing \( \forall_{\alpha*} \alpha \to \alpha \).

\( ^{2} \) For the curious, however, that is: \( S id_A id_B = idS_{AB} \) and given, additionally, \( f : A' \to A'' \in A_1 \) and \( g' : B' \to B'' \in B_1 \), then \( S (f \circ f') \circ (g \circ g') : S A'' B'' \to S A B'' \) is the composition of \( S f g' : S A' B' \to S A B'' \) and \( S f' g : S A'' B' \to S A' B'' \).
When \( A = B \), we may call \( S \) a “diagonal profunctor”. Such a thing generalizes \( \text{Hom} \) by producing “\( \text{Hom-like constructs} \)” in its target category and capturing both pre- and post-composition. The objects in the image of a profunctor are objects that can play the role of bundles of arrows of the source category.

## 4 A Particular Diagonal Profunctor

Recall, the ultimate goal is to find a categorical object (in \( \text{Set} \)) that captures the set of natural transformations between two functors \( F \) and \( G \). The two paths through a natural transformation diagram for \( \eta : F \to G \) and \( f : \tau \to \tau' \) are \((\eta \tau') \circ (Ff)\) and \((Gf) \circ (\eta \tau)\). Someone with a great deal more insight than this author realized that these are images of a particular diagonal profunctor \( S_{\text{nat},F,G} \) defined on objects by \( S_{\text{nat},F,G} \tau \tau' = \text{Hom} \tau \tau' \) and on arrows by

\[
(S_{\text{nat},F,G} (f : \tau \to \tau') (g : \tau'' \to \tau''')) \quad ((h : \text{Hom} \tau' \tau'')) = (Gg) \circ h \circ (Ff) : \text{Hom} \tau \tau'''.
\]

In particular, our two paths appear in the off-diagonal \( S_{\text{nat},F,G} \tau \tau' \) as the \( S_{\text{nat},F,G} f \) image of \( \eta \tau' \) (in \( \text{Hom} \tau' \tau' \)) and the \( S_{\text{nat},F,G} \eta \tau \) image of \( \eta \tau \) (in \( \text{Hom} \tau \tau \)).

Drawing the conclusions of the above two paragraphs at once, we have this rendering of our seemingly so simple equation \( Gf \circ (\eta \tau) = (\eta \tau') \circ Ff \) (admittedly, for all natural transformations \( \eta \)):

\[
\forall f : \tau \to \tau' \exists ! S \quad N \\
\quad \sigma \tau' : f_{\tau \to \tau'} \quad S_{\text{nat},F,G} \tau \tau' \quad S_{\text{nat},F,G} id_{\tau'} \quad S_{\text{nat},F,G} id_{\tau} \quad S_{\text{nat},F,G} \tau \tau' \quad S_{\text{nat},F,G} \tau \tau' \quad S_{\text{nat},F,G} \tau \tau' \quad S_{\text{nat},F,G} \tau \tau' \quad S_{\text{nat},F,G} \tau \tau' \quad S_{\text{nat},F,G} \tau \tau' \quad S_{\text{nat},F,G} \tau \tau'.
\]

We could imagine that there are many such \( N \) (and their associated \( \pi \) projector bundles) that made the above diagram work. If, however, there is a universal one, \( E \), such that:

\[
\forall N, \pi_N : \exists ! S \quad E \\
\quad \sigma \tau' : f_{\tau \to \tau'} \quad S_{\text{nat},F,G} \tau \tau' \quad S_{\text{nat},F,G} id_{\tau'} \quad S_{\text{nat},F,G} id_{\tau} \quad S_{\text{nat},F,G} \tau \tau' \quad S_{\text{nat},F,G} \tau \tau' \quad S_{\text{nat},F,G} \tau \tau' \quad S_{\text{nat},F,G} \tau \tau'.
\]

Then this \( E = \text{End}(S) = \int_S = \int_{\tau} S \tau \tau \) is the end of the profunctor \( S_{\text{nat},F,G} \). It represents the collection of all \( F \to G \).

## 5 Generalizing: Dinatural Transformations and Generalized Limits

We can generalize the notion of a natural transformation between two functors to a “dinatural transformation” (for “diagonally-natural”) between two diagonal profunctors. Like a natural transformation, a dinatural transformation is a quiver of arrows; possibly surprisingly, the quiver will continue to be a single-indexed object on objects of \( A \). Concretely, \( \theta : R \to S \) is a dinatural transformation if

\[
\forall f : \tau \to \tau' \exists ! S \quad R f \quad id_{\tau} \quad \theta \tau \quad S \quad \tau \quad S \quad \tau (\tau') \quad S \quad \tau (\tau').
\]

\[\text{The use of the } f \text{ symbol is unfortunate, but standard; as discussed above, the end is much more like a product than a sum. Note the direction of the arrows in its definition diagram and compare to the universal diagram defining a product!}\]
If we define a “constant bifunctor” $\kappa \tau \tau' = N$ and $\kappa f g = \text{id}_N$, then our $S_{\text{nat}, F, G}$-based rendering of natural transformations’ equation in section 4 above is a dinatural transformation $\kappa \to S_{\text{nat}, F, G}$.

Recall that a limit of a diagram $J$ included into $C$ by a full and faithful functor $F$ is the terminal object in the subcategory of $C$ whose objects $C_\kappa$ are selected by constant functors $\kappa : J \to C$ with natural transformations $\eta_\kappa : \kappa \to F$ and whose arrows $z : C_\kappa \to C_\kappa'$ indicate factoring of $\eta_\kappa \tau = \eta_\kappa' \tau \circ z$. (Phew!)

We can generalize this notion to use dinatural transformations. Given an indexing category $J$, and a diagonal profunctor $S : J \times J \to C$ we’re now interested in the subcategory of objects $C_\kappa$ picked out by constant, diagonal profunctors $\kappa$ (of the same type as $S$) which have dinatural transformations to $S(\kappa \to S)$. The terminal object in this subcategory, if there is one, is the end of $S$.

6 Preliminary Thoughts: Parametricity Without Functors

Above, we restricted our attention to the case where the type constructors $F$ and $G$ were in fact functors. We made heavy use of this fact in defining $S_{\text{nat}, F, G}$, whose End we then characterized.

We can view the natural transformation diagram as simply the assertion that $\forall \tau, \tau' : \star; f : \tau \to \tau' \exists ! \eta f : F\tau \to G\tau'$. $\eta f = \eta \tau \circ Ff = Gf \circ \eta \tau$, or, equivalently, that for each $\eta : F \to G$ there is a function, which we might also call $\eta$, of type $\forall \tau, \tau' : \star (\tau \to \tau') \to (F\tau \to G\tau')$ such that $\eta f = \eta \tau \circ Ff = Gf \circ \eta \tau$.

We could in fact take this $\eta$ as the definition of a parametric function: rather than being a type-indexed collection of functions, a parametric function is then a type-indexed collection of higher-order functions. When $F$ and $G$ are functors, we can use either to implement and adaptor function of the right type:

$\lambda \eta : \forall \beta : \star. F\beta \to G\beta \Lambda \alpha, \alpha' : \star \Lambda f : \alpha \to \alpha' \Lambda x : F\alpha (G f) (\eta \alpha x) \text{ or } \eta \alpha' ((F f) x)$ under identical binders.

If only $F$ is a functor, we could still impose an equation of the form $\eta (f : \tau \to \tau') (x : F\tau) = \eta \text{id}_{\tau'} ((F f) x)$ (at $G\tau'$). That is, the function $\eta$ cannot distinguish between being given a function $f$ and some data $x$ and being given the identity function and the data $(F f) x$.

---

I am indebted to “ski” of #haskell on Freenode for a reminder of this view and the name $\eta_f$. 