Freedom of Speech is a Comonad

Harley Eades III
Augusta University

January 11, 2016

Abstract

...

1 Introduction

The Trellys project was a large scale project to develop a general purpose dependently type functional programming language [**?**]. The most important language design decision was to have the language fragmented into two parts: a programmatic fragment and a logical fragment. The former is used to write general purpose programs, and the latter is used to verify the correctness of the programs written in the programmatic fragment. This implies that there must be a means for the logical fragment to take programs of the programmatic fragment – which we will call simply programs – in as input to be able to write predicates and proofs about those programs, but this must be done so as to not compromise consistency of the logical fragment. Thus, it must be maintained that no logical program, which we will call proofs, actually runs a diverging program. This property is called **freedom of speech**, because the logical fragment has the ability to talk about the programmatic fragment without sacrificing its integrity.

In this paper we approach this problem from a categorical perspective. Suppose we have a cartesian closed category, $\mathcal{L}$, and a cartesian closed category with fixpoints, $\mathcal{P}$. It is well-known that $\mathcal{L}$ is a model of the simply-typed λ-calculus [**?**], and $\mathcal{P}$ is a model of the simply-typed λ-calculus with the Y combinator [**?**]. The category $\mathcal{L}$ will model our logical fragment, and $\mathcal{P}$ our programmatic fragment.

Now suppose we have a symmetric monoidal adjunction $\mathcal{P} : \uparrow \dashv \downarrow : \mathcal{L}$. We can think of this adjunction as a translation between the two fragments. It is a well-known fact about adjunctions that the functor $\downarrow\uparrow : \mathcal{P} \to \mathcal{P}$ is a monad, and the functor $\uparrow\downarrow : \mathcal{L} \to \mathcal{L}$ is a comonad. The former treats termination as a monadic effect allowing the programmatic fragment to restrict itself to terminating programs, and the latter treats non-termination as a comonadic effect allowing proofs to take in as input potentially diverging programs. Notice

that the latter models freedom of speech as a comonad. It can also be shown that we have a strong monad:

$$\uparrow\downarrow A \times \downarrow\uparrow B \xrightarrow{\mathsf{st}_{A,B}} \downarrow\uparrow(\uparrow\downarrow A \times B)$$

The previous strength can be defined using the fact that we have a symmetric monoidal adjunction.

The first model of this kind is due to Benton [**?**] who described a similar situation for linear logic where $\mathcal{L}$ is a model of linear logic, and $\mathcal{P}$ is a model of propositional logic, and hence, the comonand $! : \mathcal{L} \to \mathcal{L}$ treats non-linearity as a comonadic effect, but from the perspective of $\mathcal{L}$ linearity is a monadic effect. Adjunction models like these can now be seen as beautiful models of effects [**?**].

2 The Fragments and Their Semantics

2.1 The Logical Fragment

Definition 1. *The typing relation for the* ***logical fragment*** *is defined by the following inference rules:*

$$\frac{}{\Gamma_1, x : A, \Gamma_2 \vdash_{\mathcal{L}} x : A}\ \textsc{L\_Ax} \qquad \frac{}{\Gamma \vdash_{\mathcal{L}} * : \top}\ \textsc{L\_True}$$

$$\frac{\Gamma \vdash_{\mathcal{L}} t_1 : A \quad \Gamma \vdash_{\mathcal{L}} t_2 : B}{\Gamma \vdash_{\mathcal{L}} (t_1, t_2) : A \times B}\ \textsc{L\_Prod} \qquad \frac{\Gamma \vdash_{\mathcal{L}} t : A \times B}{\Gamma \vdash_{\mathcal{L}} \mathsf{proj}_1\, t : A}\ \textsc{L\_Proj1}$$

$$\frac{\Gamma \vdash_{\mathcal{L}} t : A \times B}{\Gamma \vdash_{\mathcal{L}} \mathsf{proj}_2\, t : B}\ \textsc{L\_Proj2} \qquad \frac{\Gamma, x : A \vdash_{\mathcal{L}} t : B}{\Gamma \vdash_{\mathcal{L}} \lambda x.t : A \to B}\ \textsc{L\_Fun}$$

$$\frac{\Gamma \vdash_{\mathcal{L}} t_2 : A \quad \Gamma \vdash_{\mathcal{L}} t_1 : A \to B}{\Gamma \vdash_{\mathcal{L}} t_1\, t_2 : B}\ \textsc{L\_App}$$

Terms in context are defined by the following inference rules:

$$\frac{}{\Gamma_1, x : A, \Gamma_2 \vdash_{\mathcal{L}} x = x : A}\ \textsc{Leq\_Ax} \qquad \frac{\Gamma \vdash_{\mathcal{L}} t : \top}{\Gamma \vdash_{\mathcal{L}} t = * : \top}\ \textsc{Leq\_Unit}$$

$$\frac{\Gamma \vdash_{\mathcal{L}} t_1 : A \quad \Gamma \vdash_{\mathcal{L}} t_2 : B}{\Gamma \vdash_{\mathcal{L}} \mathsf{proj}_1\, t = t_1 : A}\ \textsc{Leq\_Proj1} \qquad \frac{\Gamma \vdash_{\mathcal{L}} t_1 : A \quad \Gamma \vdash_{\mathcal{L}} t_2 : B}{\Gamma \vdash_{\mathcal{L}} \mathsf{proj}_2\, t = t_2 : B}\ \textsc{Leq\_Proj2}$$

$$\frac{\Gamma \vdash_{\mathcal{L}} t : A \times B}{\Gamma \vdash_{\mathcal{L}} (\mathsf{proj}_1\, t, \mathsf{proj}_2\, t) = t : A \times B}\ \textsc{Leq\_EtaP}$$

$$\frac{\Gamma \vdash_{\mathcal{L}} t = t' : A \times B}{\Gamma \vdash_{\mathcal{L}} \mathsf{proj}_1\, t = \mathsf{proj}_1\, t : A \times B}\ \textsc{Leq\_Proj1C}$$

$$\frac{\Gamma \vdash_{\mathcal{L}} t = t' : A \times B}{\Gamma \vdash_{\mathcal{L}} \mathsf{proj}_2\, t = \mathsf{proj}_2\, t : A \times B}\ \textsc{Leq\_Proj2C}$$

$$\frac{\Gamma \vdash_{\mathcal{L}} t_1 = t_1' : A \quad \Gamma \vdash_{\mathcal{L}} t_2 = t_2' : B}{\Gamma \vdash_{\mathcal{L}} (t_1, t_2) = (t_1, t_2') : A \times B}\ \textsc{Leq\_PairC}$$

$$\frac{\begin{array}{c}\Gamma \vdash_{\mathcal{L}} t' : B \\ \Gamma, x : A \vdash_{\mathcal{L}} t : B\end{array}}{\Gamma \vdash_{\mathcal{L}} (\lambda x.t)\, t' = [t'/x]t : B}\ \textsc{Leq\_Beta} \qquad \frac{\Gamma \vdash_{\mathcal{L}} t : A \to B \quad x \notin \mathsf{FV}(t)}{\Gamma \vdash_{\mathcal{L}} (\lambda x.t\, x) = t : A \to B}\ \textsc{Leq\_Eta}$$

$$\frac{\Gamma, x : A \vdash_{\mathcal{L}} t = t' : B}{\Gamma \vdash_{\mathcal{L}} \lambda x.t = \lambda x.t' : B}\ \textsc{Leq\_FunC} \qquad \frac{\Gamma \vdash_{\mathcal{L}} t : A \quad \Gamma, x : A \vdash_{\mathcal{L}} t_1 = t_2 : B}{\Gamma \vdash_{\mathcal{L}} [t/x]t_1 = [t/x]t_2 : B}\ \textsc{Leq\_Subst}$$

$$\frac{\Gamma \vdash_{\mathcal{L}} t_1 = t_2 : A}{\Gamma, \Gamma' \vdash_{\mathcal{L}} t_1 = t_2 : A}\ \textsc{Leq\_Weak} \qquad \frac{\Gamma_1, x : A, y : B, \Gamma_2 \vdash_{\mathcal{L}} t_1 = t_2 : C}{\Gamma_1, y : B, x : A, \Gamma_2 \vdash_{\mathcal{L}} t_1 = t_2 : C}\ \textsc{Leq\_Ex}$$

$$\frac{\Gamma \vdash_{\mathcal{L}} t : A}{\Gamma \vdash_{\mathcal{L}} t = t : A}\ \textsc{Leq\_Refl} \qquad \frac{\Gamma \vdash_{\mathcal{L}} t_1 = t_2 : A}{\Gamma \vdash_{\mathcal{L}} t_2 = t_1 : A}\ \textsc{Leq\_Sym}$$

$$\frac{\Gamma \vdash_{\mathcal{L}} t_1 = t_2 : A \quad \Gamma \vdash_{\mathcal{L}} t_2 = t_3 : A}{\Gamma \vdash_{\mathcal{L}} t_1 = t_3 : A}\ \textsc{Leq\_Trans}$$

2.2 The Programmatic Fragment

Definition 2. *The typing relation for the* ***programmatic fragment*** *is defined by the following inference rules:*

$$\frac{}{\Gamma_1, x : X, \Gamma_2 \vdash_{\mathcal{P}} x : X}\ \text{P\_Ax} \qquad \frac{}{\Gamma \vdash_{\mathcal{P}} * : \top}\ \text{P\_Unit} \qquad \frac{}{\Gamma \vdash_{\mathcal{P}} 0 : \mathsf{Nat}}\ \text{P\_Zero}$$

$$\frac{}{\Gamma \vdash_{\mathcal{P}} \mathbf{suc} : \mathsf{Nat} \to \mathsf{Nat}}\ \text{P\_Suc} \qquad \frac{}{\Gamma \vdash_{\mathcal{P}} \mathsf{fix} : (X \to X) \to X}\ \text{P\_Fix}$$

$$\frac{\Gamma \vdash_{\mathcal{P}} t_1 : X \quad \Gamma \vdash_{\mathcal{P}} t_2 : Y}{\Gamma \vdash_{\mathcal{P}} (t_1, t_2) : X \times Y}\ \text{P\_Prod} \qquad \frac{\Gamma \vdash_{\mathcal{P}} t : X \times Y}{\Gamma \vdash_{\mathcal{P}} \mathsf{proj}_1\, t : X}\ \text{P\_Proj1}$$

$$\frac{\Gamma \vdash_{\mathcal{P}} t : X \times Y}{\Gamma \vdash_{\mathcal{P}} \mathsf{proj}_2\, t : X}\ \text{P\_Proj2} \qquad \frac{\Gamma, x : X \vdash_{\mathcal{P}} t : Y}{\Gamma \vdash_{\mathcal{P}} \lambda x.t : X \to Y}\ \text{P\_Fun}$$

$$\frac{\Gamma \vdash_{\mathcal{P}} t_2 : X \quad \Gamma \vdash_{\mathcal{P}} t_1 : X \to Y}{\Gamma \vdash_{\mathcal{P}} t_1\, t_2 : Y}\ \text{P\_App} \qquad \frac{\Gamma \vdash_{\mathcal{P}} t : \mathsf{Nat} \quad \Gamma \vdash_{\mathcal{P}} t_1 : X \quad \Gamma, x : \mathsf{Nat} \vdash_{\mathcal{P}} t_2 : X}{\Gamma \vdash_{\mathcal{P}} \mathsf{case}\, t\, \{0 \to t_1, \mathbf{suc}\, x \to t_2\} : X}\ \text{P\_Case}$$

Terms in context are defined by the following inference rules:

$$\frac{}{\Gamma_1, x : X, \Gamma_2 \vdash_{\mathcal{P}} x = x : X} \text{ PEq\_Ax} \qquad \frac{\Gamma \vdash_{\mathcal{P}} t : \top}{\Gamma \vdash_{\mathcal{P}} t = * : \top} \text{ PEq\_Unit}$$

$$\frac{\Gamma \vdash_{\mathcal{P}} t_1 : X \quad \Gamma \vdash_{\mathcal{P}} t_2 : Y}{\Gamma \vdash_{\mathcal{P}} \mathsf{proj}_1\, t = t_1 : X} \text{ PEq\_Proj1} \qquad \frac{\Gamma \vdash_{\mathcal{P}} t_1 : X \quad \Gamma \vdash_{\mathcal{P}} t_2 : Y}{\Gamma \vdash_{\mathcal{P}} \mathsf{proj}_2\, t = t_2 : Y} \text{ PEq\_Proj2}$$

$$\frac{\Gamma \vdash_{\mathcal{P}} t : X \times Y}{\Gamma \vdash_{\mathcal{P}} (\mathsf{proj}_1\, t, \mathsf{proj}_2\, t) = t : X \times Y} \text{ PEq\_EtaP}$$

$$\frac{\Gamma \vdash_{\mathcal{P}} t = t' : X \times Y}{\Gamma \vdash_{\mathcal{P}} \mathsf{proj}_1\, t = \mathsf{proj}_1\, t : X \times Y} \text{ PEq\_Proj1C}$$

$$\frac{\Gamma \vdash_{\mathcal{P}} t = t' : X \times Y}{\Gamma \vdash_{\mathcal{P}} \mathsf{proj}_2\, t = \mathsf{proj}_2\, t : X \times Y} \text{ PEq\_Proj2C}$$

$$\frac{\Gamma \vdash_{\mathcal{P}} t_1 = t_1' : X \quad \Gamma \vdash_{\mathcal{P}} t_2 = t_2' : Y}{\Gamma \vdash_{\mathcal{P}} (t_1, t_2) = (t_1, t_2') : X \times Y} \text{ PEq\_PairC}$$

$$\frac{\begin{array}{c}\Gamma \vdash_{\mathcal{P}} t' : Y \\ \Gamma, x : X \vdash_{\mathcal{P}} t : Y\end{array}}{\Gamma \vdash_{\mathcal{P}} (\lambda x.t)\, t' = [t'/x]t : Y} \text{ PEq\_Beta} \qquad \frac{\Gamma \vdash_{\mathcal{P}} t : X \to Y \quad x \notin \mathsf{FV}(t)}{\Gamma \vdash_{\mathcal{P}} (\lambda x.t\, x) = t : X \to Y} \text{ PEq\_Eta}$$

$$\frac{\Gamma, x : X \vdash_{\mathcal{P}} t = t' : Y}{\Gamma \vdash_{\mathcal{P}} \lambda x.t = \lambda x.t' : Y} \text{ PEq\_FunC} \qquad \frac{\Gamma \vdash_{\mathcal{P}} t : X \to X}{\Gamma \vdash_{\mathcal{P}} \mathsf{fix}\, t = t\, (\mathsf{fix}\, t) : X} \text{ PEq\_Fix}$$

$$\frac{\Gamma \vdash_{\mathcal{P}} t = t' : X \to X}{\Gamma \vdash_{\mathcal{P}} \mathsf{fix}\, t = \mathsf{fix}\, t' : X} \text{ PEq\_FixC} \qquad \frac{\Gamma \vdash_{\mathcal{P}} t = t' : \mathsf{Nat}}{\Gamma \vdash_{\mathcal{P}} \mathbf{suc}\, t = \mathbf{suc}\, t' : \mathsf{Nat}} \text{ PEq\_Suc}$$

$$\frac{\begin{array}{c}\Gamma \vdash_{\mathcal{P}} t = 0 : \mathsf{Nat} \\ \Gamma \vdash_{\mathcal{P}} t_1 : X \quad \Gamma, x : \mathsf{Nat} \vdash_{\mathcal{P}} t_2 : X\end{array}}{\Gamma \vdash_{\mathcal{P}} \mathsf{case}\, t\, \{0 \to t_1, \mathbf{suc}\, x \to t_2\} = t_1 : X} \text{ PEq\_CaseB}$$

$$\frac{\begin{array}{c}\Gamma \vdash_{\mathcal{P}} t = \mathbf{suc}\, n : \mathsf{Nat} \\ \Gamma \vdash_{\mathcal{P}} t_1 : X \quad \Gamma, x : \mathsf{Nat} \vdash_{\mathcal{P}} t_2 : X\end{array}}{\Gamma \vdash_{\mathcal{P}} \mathsf{case}\, t\, \{0 \to t_1, \mathbf{suc}\, x \to t_2\} = [n/x]t_2 : X} \text{ PEq\_CaseS}$$

$$\frac{\begin{array}{c}\Gamma \vdash_{\mathcal{P}} t_1 = t_1' : X \\ \Gamma, x : \mathsf{Nat} \vdash_{\mathcal{P}} t_2 = t_2' : X \quad \Gamma \vdash_{\mathcal{P}} t = t' : \mathsf{Nat}\end{array}}{\Gamma \vdash_{\mathcal{P}} \mathsf{case}\, t\, \{0 \to t_1, \mathbf{suc}\, x \to t_2\} = \mathsf{case}\, t'\, \{0 \to t_1', \mathbf{suc}\, x \to t_2'\} : X} \text{ PEq\_CaseC}$$

$$\frac{\Gamma \vdash_{\mathcal{P}} t : X \quad \Gamma, x : X \vdash_{\mathcal{P}} t_1 = t_2 : Y}{\Gamma \vdash_{\mathcal{P}} [t/x]t_1 = [t/x]t_2 : Y} \text{ PEq\_Subst} \qquad \frac{\Gamma \vdash_{\mathcal{P}} t_1 = t_2 : X}{\Gamma, \Gamma' \vdash_{\mathcal{P}} t_1 = t_2 : X} \text{ PEq\_Weak}$$

$$\frac{\Gamma_1, x : X, y : Y, \Gamma_2 \vdash_{\mathcal{P}} t_1 = t_2 : Z}{\Gamma_1, y : Y, x : X, \Gamma_2 \vdash_{\mathcal{P}} t_1 = t_2 : Z} \text{ PEq\_Ex} \qquad \frac{\Gamma \vdash_{\mathcal{P}} t : X}{\Gamma \vdash_{\mathcal{P}} t = t : X} \text{ PEq\_Refl}$$

$$\frac{\Gamma \vdash_{\mathcal{P}} t_1 = t_2 : X}{\Gamma \vdash_{\mathcal{P}} t_2 = t_1 : X} \text{ PEq\_Sym} \qquad \frac{\Gamma \vdash_{\mathcal{P}} t_1 = t_2 : X \quad \Gamma \vdash_{\mathcal{P}} t_2 = t_3 : X}{\Gamma \vdash_{\mathcal{P}} t_1 = t_3 : X} \text{ PEq\_Trans}$$

Example 3. *In this example we give a few examples of programs that cannot be written in the logical fragment:*

1. *The following program diverges:*

$$\mathsf{fix}\,\mathbf{suc}$$

   In fact,

$$\mathsf{fix}\,\mathbf{suc} \rightsquigarrow \mathbf{suc}\,(\mathsf{fix}\,\mathbf{suc}) \rightsquigarrow \mathbf{suc}\,(\mathbf{suc}\,(\mathsf{fix}\,\mathbf{suc})) \rightsquigarrow \cdots$$

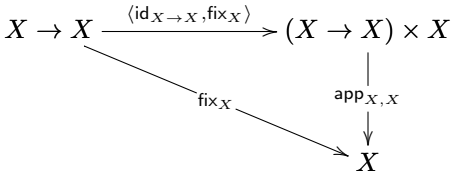
2. *The following program defines natural number addition:*

$$\mathsf{fix}\,(\lambda r.\lambda n_1.\lambda n_2.\mathsf{case}\,n_2\,\{0 \to n_1, \mathbf{suc}\,x \to \mathbf{suc}\,(r\,n_1\,x)\})$$

The most interesting part of the programmatic fragment is the way in which natural numbers are represented. The mixture of fix, which is the Y-combinator, and case provides the calculus with the ability to write both terminating and non-terminating programs on the natural numbers.

The question now becomes how do we model fix and case categorically? To model fix we endow a cartesian closed category with fixpoints [**?**].

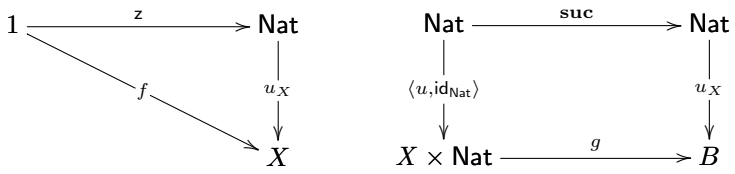
Definition 4. *Suppose $\mathcal{C}$ is a cartesian closed category. Then we say $\mathcal{C}$ **has fixpoints** if for any object X of $\mathcal{C}$, there is a morphism $\mathsf{fix}_X : (X \to X) \longrightarrow X$ such that the following diagram commutes:*



The commuting diagram in the above definition models the reduction rule for fix, and so it is pretty easy to see that this definition will allow us to model fix categorically.

Perhaps more interestingly we use a novel approach to modeling natural numbers including case.

Definition 5. *Suppose $\mathcal{C}$ is a cartesian closed category. A **Scott natural number object (SNNO)** is an object Nat of $\mathcal{C}$ and morphisms $\mathsf{z} : 1 \longrightarrow \mathsf{Nat}$ and $\mathbf{suc} : \mathsf{Nat} \longrightarrow \mathsf{Nat}$ of $\mathcal{C}$, such that, for any morphisms $f : 1 \longrightarrow X$ and $g : X \times \mathsf{Nat} \longrightarrow X$ of $\mathcal{C}$ there is a unique morphism $u_X : \mathsf{Nat} \longrightarrow X$ making the following diagrams commute:*



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

$$\begin{aligned} u\,\mathsf{z} &= f \\ u\,(\mathbf{suc}\,x) &= g\,(u\,x)\,x \end{aligned}$$

This formalization of natural numbers is inspired by the definition of Scott Numerals [**?**] where the notion of a case distinction is built into the encoding. SNNOs differ from the more traditional natural number objects [**?**] in the type of $g : X \times \mathsf{Nat} \longrightarrow X$, which is traditionally, $g : X \longrightarrow X$, and hence, g will only receive as input the recursive call. This definition is too restrictive for the situation here, because to model case we need the ability to obtain the predecessor.

We now have all the necessary structure to describe the model of the programmatic fragment.

Definition 6. *A model of the programmatic fragment, called a* $\mathcal{P}$***-model**, consists of a cartesian closed category,* $\mathcal{C}$*, with fixpoints and a SNNO.*

Finally, we show the following.

Theorem 7 (Soundness)**.** *Suppose* $\mathcal{P}$ *is a* $\mathcal{P}$*-model, and for each type* X *there is an object* $[\![X]\!]$ *of* $\mathcal{C}$*. If* $\Gamma \vdash_{\mathcal{P}} t = t' : X$*, then there exists morphisms* $[\![t]\!] : [\![\Gamma]\!] \longrightarrow [\![X]\!]$ *and* $[\![t']\!] : [\![\Gamma]\!] \longrightarrow [\![X]\!]$ *of* $\mathcal{P}$*, such that,* $[\![t]\!] = [\![t']\!]$*.*

3 Related Work

Write

4 Conclusion

Write

References