Adjoint Programming: Termination as an Effect

Harley Eades III
Augusta University

January 25, 2016

Abstract

...

1 Introduction

It is a well-known fact about adjunctions that they induce a monoad/comonad pair on some category. Recently, adjunctions have been proposed as an elegant model for resources and effects [**?**]. Even more recently, they have been proposed as a model of constructive modal logic. However, is it worthwhile to adopt adjunctions directly into a programming language? If they are adopted, then how do we program with them? In this paper we construct a typed λ-calculus extended with an adjoint pair of functors with a focus on programming. Instead of working in completely generality we instead focus on a single concrete, but interesting, example.

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. On the other side of the coin the programmatic fragment should have the ability to construct programs using the logical fragment. This property is called **freedom of speech**, because both fragments have the ability to talk about each other without sacrificing their integrity.

The running example of this paper is to 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 $\mathcal{P} : T \dashv D : \mathcal{L}$. We can think of this adjunction as a translation between the two fragments. It is a well-known fact about adjunctions that the functor $\downarrow = DT : \mathcal{P} \to \mathcal{P}$ is a comonad, and the functor $\uparrow = TD : \mathcal{L} \to \mathcal{L}$ is a monad. The former treats termination as a comonadic effect allowing the programmatic fragment to restrict itself to terminating programs, and the latter treats non-termination as a monadic effect allowing proofs to take in as input potentially diverging programs. Thus, we can see freedom of speech as an adjoint situation. It can also be shown that we have a strong monad:

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

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

It does make a lot of since treating termination as a comonad. Every comonad is equipped with an operation $\varepsilon_X : \downarrow X \to X$. We can view this comonad as transportation of terminating programs to non-terminating programs. That is, any terminating program is a perfectly good diverging program. This is one property the Trellys project had difficulties accommodating.

One application where this is useful is in programming languages such as Haskell where programmers by default program in potentially diverging setting. However, to gain assurances it is often preferred to be able to pick and choose which programs one would want to be terminating, and have it enforced by the type checker. A termination comonad would allow the programmer to develop their program under the comonad to insure termination.

We will construct typed λ-calculi corresponding to both $\mathcal{L}$ and $\mathcal{P}$ respectively. Then we will bring these two calculi together through a syntactic adjunction. Then we will consider various programmatic questions. For example, every program of type $\downarrow X$ should terminate, but proofs of type $\uparrow A$ should be allowed to diverge. In addition to these syntactic questions we discuss novel contributions of these adjoint categorical models.

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

2 The Fragments and Their Semantics

We first must define the two fragments and give their semantics. The logic fragment will be kept simple and amounts to the simply-typed λ-calculus, but the programmatic fragment will be slightly more interesting. Since we are interested in programming it will contain the natural numbers, with a natural number eliminator in the form of a case analysis, and the Y-combinator. Each of the theories will be presented using the style of Crole [**?**].

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

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

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

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

Figure 1: The Logical Fragment: Typing Relation

2.1 The Logical Fragment

The logical fragment is defined in Figure 1 and Figure 2. The following is a well-known fact [**?**].

Lemma 1. *The logical fragment has a sound and complete interpretation into cartesian closed categories.*

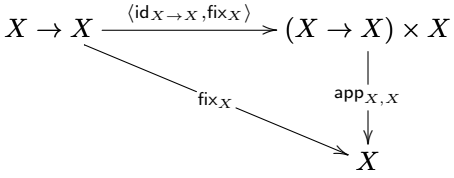
2.2 The Programmatic Fragment

The programmatic fragment is a bit more interesting, and corresponds to the simply-typed λ-calculus with natural numbers and the Y-combinator. The typing relation is defined in Figure 3, and the terms-in-context inference rules are defined in Figure 4. The natural number eliminator, case, can be mixed with the Y-combinator to write generally recursive programs on the natural numbers. For example, the following 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 question now becomes how do we model fix and case categorically? To model fix we endow a cartesian closed category with fixpoints [**?**].

Definition 2. *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:*



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

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

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

$$\frac{\Gamma \vdash_{\mathcal{L}} t = t' : A \times B}{\Gamma \vdash_{\mathcal{L}} \mathsf{proj}_1\, t = \mathsf{proj}_1\, t' : A}\ \textsc{Leq\_Proj1C} \qquad \frac{\Gamma \vdash_{\mathcal{L}} t = t' : A \times B}{\Gamma \vdash_{\mathcal{L}} \mathsf{proj}_2\, t = \mathsf{proj}_2\, t' : B}\ \textsc{Leq\_Proj2C}$$

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

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

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

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

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

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

Figure 2: The Logical Fragment: Equality

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

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

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

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

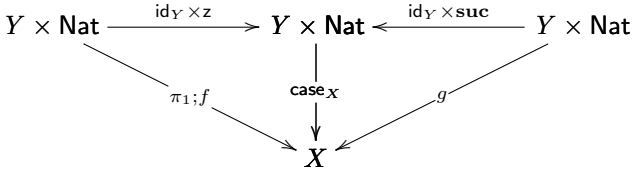
$$\frac{\Gamma \vdash_{\mathcal{P}} t_2 : X \quad \Gamma \vdash_{\mathcal{P}} t_1 : X \to Y}{\Gamma \vdash_{\mathcal{P}} t_1\, t_2 : Y}\ \textsc{P\_App} \qquad \frac{\Gamma \vdash_{\mathcal{P}} t : \mathsf{Nat} \quad \Gamma \vdash_{\mathcal{P}} t_1 : X \quad \Gamma, x : \mathsf{Nat} \vdash_{\mathcal{P}}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

Figure 3: Programmatic Fragment: Typing Relation

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 with their eliminator.

Definition 3. *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 $f : Y \longrightarrow X$ and $g : Y \times \mathsf{Nat} \longrightarrow X$ of $\mathcal{C}$ there is a unique morphism $\mathsf{case}_X : Y \times \mathsf{Nat} \longrightarrow X$ making the following diagrams commute:*



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

$$\begin{aligned} u\, y\, 0 &= f\, y \\ u\, y\, (\mathbf{suc}\, x) &= g\, y\, x \end{aligned}$$

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. We can think of Y in the source object of case as the type additional inputs that will be passed to both f and g, but we can think of Nat in the source object of case as the type of the scrutiny. Thus, since in the base case there is no predecessor, f, will not require the scrutiny, and so it is ignored.

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

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

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

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

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

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

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

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

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

$$\frac{\Gamma \vdash_{\mathcal{P}} t_1 : X \quad \Gamma, x : \mathsf{Nat} \vdash_{\mathcal{P}} t_2 : X}{\Gamma \vdash_{\mathcal{P}} \mathsf{case}\, 0\, \{0 \to t_1, \mathbf{suc}\, x \to t_2\} = t_1 : X}\ \textsc{PEq\_CaseB}$$

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

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

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

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

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

Figure 4: Programmatic Fragment: Equality

One major difference between SNNOs and the more traditional natural number objects is that in the definition of the latter g is defined by well-founded recursion. However, SNNOs do not allow this, but in the presence of fixpoints we are able to regain this feature without having to bake it into the definition of natural number objects. However, to allow this we have found that when combining fixpoints and case analysis to define terminating functions on the natural numbers it is necessary to uniformly construct the input to both f and g due to the reduction rule of fix. Thus, we extend the type of f to $Y \times \mathsf{Nat}$, but then ignore the second projection when reaching the base case.

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

Definition 4. *A model of the programmatic fragment, called a $\mathcal{P}$-**model**, consists of a cartesian closed category, $\mathcal{C}$, with fixpoints and a SNNO.*

The following definition gives an interpretation of types and typing contexts into a $\mathcal{P}$-model.

Definition 5. *Suppose $\mathcal{P}$ is a $\mathcal{P}$-model.*

- *The interpretation of types as objects is as follows:*

$$\begin{array}{lcl} [\![\top]\!] & = & 1 \\ [\![\mathsf{Nat}]\!] & = & \mathsf{Nat} \\ [\![X \times Y]\!] & = & [\![X]\!] \times [\![Y]\!] \\ [\![X \to Y]\!] & = & [\![X]\!] \to [\![Y]\!] \end{array}$$

- *The interpretation of typing contexts is as follows:*

$$\begin{array}{lcl} [\![\cdot]\!] & = & 1 \\ [\![\Gamma, x_1 : X_1]\!] & = & [\![\Gamma]\!] \times [\![X_1]\!] \end{array}$$

We always assume that the interpretation of typing contexts is right-associated.

The following result is mostly known, but we give the cases of the proof that are most interesting. The cases for SNNOs are novel, but straightforward given their definition.

Theorem 6 (Soundness)**.** *Suppose $\mathcal{P}$ is a $\mathcal{P}$-model.*

i. If $\Gamma \vdash_{\mathcal{P}} t : X$, then there is a morphism, $[\![t]\!] : [\![\Gamma]\!] \longrightarrow [\![X]\!]$, in $\mathcal{P}$, and

ii. If $\Gamma \vdash_{\mathcal{P}} t = t' : X$, then $[\![t]\!] = [\![t']\!]$ in $\mathcal{P}$.

Proof. We will prove part two explicitly which depends on part one in such away that proving part one and then part two would require a lot of repeated constructions. We proceed by induction on the form of the assumed term-in-context derivation, but since the majority of the programmatic fragment is well-known to have a sound and complete interpretation into a cartesian closed category we only show the one of the cases for natural numbers.

Case.

$$\frac{\begin{array}{l}\Gamma \vdash_{\mathcal{P}} n : \mathsf{Nat} \\ \Gamma \vdash_{\mathcal{P}} t_1 : X \quad \Gamma, x : \mathsf{Nat} \vdash_{\mathcal{P}} t_2 : X\end{array}}{\Gamma \vdash_{\mathcal{P}} \mathsf{case}\,(\mathbf{suc}\, n)\,\{0 \to t_1, \mathbf{suc}\, x \to t_2\} = [n/x]t_2 : X}\ \textsc{CaseS}$$

We have the following morphisms by part one:

$$[\![\Gamma]\!] \xrightarrow{[\![n]\!]} \mathsf{Nat}$$

$$[\![\Gamma]\!] \xrightarrow{[\![t_1]\!]} [\![X]\!]$$

$$[\![\Gamma]\!] \times \mathsf{Nat} \xrightarrow{[\![t_2]\!]} [\![X]\!]$$

Then by the definition of SNNO we know there exists an unique morphism

$$Y \times \mathsf{Nat} \xrightarrow{\mathsf{case}_{[\![X]\!]}} [\![X]\!]$$

such that the following is one equation that holds:

$$(\mathsf{id}_{[\![\Gamma]\!]} \times \mathsf{suc}); \mathsf{case}_{[\![X]\!]} = [\![t_2]\!]$$

We also have the following interpretation for any $[\![\Gamma \vdash_{\mathcal{P}} t : \mathsf{Nat}]\!]$:

$$[\![\mathsf{case}\, t\, \{0 \to t_1, \mathbf{suc}\, x \to t_2\}]\!] = \langle \mathsf{id}_{[\![\Gamma]\!]}, [\![t]\!] \rangle; \mathsf{case}_{[\![X]\!]} : [\![\Gamma]\!] \to [\![X]\!]$$

We obtain our result because $[\![\Gamma \vdash_{\mathcal{P}} \mathbf{suc}\, n : \mathsf{Nat}]\!] = [\![n]\!]; \mathsf{suc}$:

$$\begin{array}{rcl}\langle \mathsf{id}_{[\![\Gamma]\!]}, [\![n]\!]; \mathsf{suc} \rangle; \mathsf{case}_{[\![X]\!]} & = & \langle \mathsf{id}_{[\![\Gamma]\!]}, [\![n]\!] \rangle; (\mathsf{id}_{[\![\Gamma]\!]} \times \mathsf{suc}); \mathsf{case}_{[\![X]\!]} \\ & = & \langle \mathsf{id}_{[\![\Gamma]\!]}, [\![n]\!] \rangle; [\![t_2]\!].\end{array}$$

□

3 A Termination/Non-Termination Adjoint Model

We have so far laid out both the syntactic and categorical formalizations of the logical fragment and the programmatic fragment. At this point we turn to joining these two fragments together using an adjunction, but before this we must acquaint ourselves with the basic notions involved in the construction of such a model.

3.1 The Basic Tools

We have been working in cartesian closed categories, but much of the basic categorical tools we will employ arose from the study of symmetric monoidal closed categories which are at the heart of categorical models of linear logic.

4 Related Work

Write

5 Conclusion

Write

References