慣性ドリフト: From 0 to Normalization by Gluing in 4.9 seconds
A Brisk Drift through Categorical Semantics of Lambda Calculi

---

A Thesis
Presented to
The Established Interdisciplinary Committee for Mathematics and Computer Science Mathematics and Natural Sciences
Reed College

In Partial Fulfillment
of the Requirements for the Degree
Bachelor of Arts

Jay Kruer

December 2021

Approved for the Division
(Mathematics)

Angélica Osorno	James (Jim) Fix

Acknowledgements

Preface

> And further, by these, my son, be admonished: of making many books there is no end; and much study is a weariness of the flesh.
>
> Ecclesiastes 12:12, KJV.

I don't know what to put here, but this is certainly a funny quote that should find a home somewhere in the thesis.

Table of Contents

Introduction **1**

Chapter 1: Functorial semantics: sketches and their models; algebraic theories and their algebras **3**
1.1 Elementary sketches and their models 4
1.1.1 An algebraic prelude 4
1.1.2 Elementary sketches 4
1.1.3 The category of contexts and substitutions 7
1.1.4 Models are essentially set-valued functors out of the category of a sketch 8
1.2 Algebraic theories and their algebras 9

Chapter 2: Functorial semantics of the simply typed lambda calculus in cartesian closed categories **15**
2.1 The simply typed lambda calculus 15
2.2 Lambda theories, beta-eta rules, and cartesian closed categories . . . 16
2.2.1 An algebraic theory of the lambda calculus 17

Chapter 3: Normalization by evaluation: the classical perspective . . **21**

Chapter 4: Normalization by gluing **25**
4.1 Variable-arity Kripke relations 26
4.2 The comma construction and friends 26
4.2.1 An easy comma: the category of renamings 26
4.2.2 A harder example: the gluing category; or, the category of proof-relevant Ren-Kripke relations over types 27
4.3 Stratified syntax, theories, presheaves, algebras 30
4.3.1 Variables and binding in Renhat 30
4.3.2 Stratifying neutral and normal terms 31
4.3.3 Presheaves of syntax over the category of renamings 33
4.3.4 A stratified lambda-algebra of neutrals and normals in Renhat 34
4.4 Onward! 34
4.4.1 Cartesian closed structure for the gluing category 34
4.4.2 Gluing syntax to semantics, and a glued algebra 38
4.4.3 Taking stock: charting a path to normalization 40

4.4.4 Reify-reflect yoga 41
4.4.5 A promise kept: a normalization function 42
4.4.6 Reaping what we've sown: easy proofs of correctness properties 42

Conclusion **45**

List of Tables

List of Figures

Abstract

Normalization by gluing is based.

Introduction

> You don't need a weatherman to
> know which way the wind blows.
>
> Bob Dylan

Blah blah blah, I should write some stuff about how category theory helps us avoid insane proof heuristics in metatheory of type theory.

Chapter 1

Functorial semantics: sketches and their models; algebraic theories and their algebras

In this chapter we develop tools for reasoning about (syntactic) *theories*, which are in some sense "notions of" abstract structure. Examples of theories include the theory of rings, and simple type theory. To discuss how we write down our theories, we need another level of abstraction. We will work with several *notions of* (syntactic) theory, and we will more laconically refer to a notion of syntactic theory as a *doctrine*. A *doctrine* is something like a meta-framework specifying how we are to write down a theory. The first doctrine we will consider is that of the *elementary sketch*. The doctrine of the *elementary sketch* allows us to write down theories involving *unary* operations (those defined over a single argument.) This restriction on the arity of operations turns out to be quite limiting. To address this, we will later upgrade the doctrine of elementary sketches to the doctrine of *algebraic theories* which allow encoding operations with any finite number of arguments, thus covering a broad variety of theories. Algebraic theories are known more famously as *Lawvere theories* after categorical logic superstar (and former Reed College professor!) William Lawvere who originally studied them while building his functorial treatment of universal algebra. Keeping with our sketchy terminology and emphasizing the doctrinal upgrade, algebraic theories are also called *finite product sketches*. As an example of the strength of algebraic theories, we will show how to write down (as an algebraic theory) what it means to be a ring, with no reference to sets or functions. In the following chapter, we will use the doctrine of algebraic theories to develop categorical semantics of the lambda calculus. This chapter is strictly expository in nature. Much of the following presentation draws heavily from Paul Taylor's *Practical Foundations of Mathematics* [**taylor_practical_1999**]. Our humble contribution is to flesh out some of his examples, add some of our own, and make parts of the presentation more palatable and quickly digestible to the reader already acquainted with basic category theory and type theory.

1.1 Elementary sketches and their models

1.1.1 An algebraic prelude

We begin by recalling from algebra the notion of an *action* of, say, a group or a monoid. Actions are, from our perspective, a way of giving meaning, or *semantics* to elements of a set which enjoys some algebraic structure.

Definition 1. Recall that a **covariant action** of a group or monoid $(M, id, \cdot)$ on a set A is a binary operation $(-)_* (=) : M \times A \to A$ such that $\mathrm{id}_* a = a$ and $(g \circ f)_* a = g_* (f_* a)$. We can similarly a define the notion of a **contravariant action** which similarly requires the identity action to do nothing, but instead flips the order of action for compositions: $(g \circ f)^* a = f^* (g^* a)$

For example, in algebra we learn that the dihedral group of order 8, written D_4, acts on the square (with uniquely identified points) by reflections and rotations[1]; each of the operations encoded by the action results in the same image of the square up to ignoring the unique identity of the points we started with. This action gives geometric meaning to each of the group elements, and was used in the first day of the author's algebra class to explain the algebraic mechanics of the group itself; discovering which elements of the group are inverse to one another is done by geometric experimentation using a square with uniquely colored vertices. Similarly, the symmetric groups S_n act on lists of length n by permutation of the list elements. In this case, the action can be even more trivially defined. We now turn to the definition of an important property of actions: *faithfulness*.

Definition 2. A **faithful** action $(-)_*$ is one for which things are *semantically* equal (or, act the same) only when they are *syntactically* equal (or, *are* the same as far as your eyeballs are concerned.) More precisely rendered, we require:

$$\forall (a : A) . f_* a = g_* a \implies f = g$$

It can now be seen that the crucial property enjoyed by the natural action of D_8 on the square which enabled our use of paper cutouts in studying the group is faithfulness. If the action were not faithful, determining which operations in D_8 are inverses would not be so easy as printing out a square and plugging away, because we may (among other catastrophes) be working with an action which may not take only the identity element to the leave-everything-in-place operation on the square.

Having gesticulated that actions gives groups and monoids their meaning, we turn to the development of the doctrines of *elementary sketch* and *algebraic theory* which will allow us to generalize both sets with algebraic structures and their actions to new settings.

1.1.2 Elementary sketches

As promised, we begin with the definition.

[1] https://groupprops.subwiki.org/wiki/Dihedral_group:D8

Definition 3. An **elementary sketch** is comprised of the following data:

1. a collection $X, Y, Z, ...$ of named **base types** or **sorts**
2. a **variable** $x : X$ for each occurrence of each named sort.
3. a collection of **unary operation-symbols** or **constructors** τ having at most one variable. As a clarifying example: when sketching type theories, we will write $x : X \vdash \tau(x) : Y$.
4. a collection of equations or **laws** of the form:

$$\tau_n \left(\tau_{n-1} \left(\cdots \tau_2 \left(\tau_1 \left(x\right)\right) \cdots\right)\right) = \sigma_m \left(\sigma_{m-1} \left(\cdots \sigma_2 \left(\sigma_1 \left(x\right)\right) \cdots\right)\right)$$

We will discuss the generality provided by this definition after some intervening examples. One of the simplest examples is the sketch of a (free) monoid on some set S:

Example 1 (Sketch of a (free) monoid)**.** The requisite data for the sketch are as follows:

1. The collection of sorts is the singleton $\{M\}$.
2. The collection of variables is $\{m : M\}$.
3. The collection of operation symbols is the set S. Each has as its signature $M \to M$
4. No equations are imposed.

To really buy that this sketch generates a free monoid, we need an intervening definition of a concept we will get a lot of mileage out of in this thesis. The idea should already be familiar from our study of type theory, despite the drastically simplified setting.

Definition 4. Given an elementary sketch, a **term** $x : \Gamma \vdash X$ is a string of composable unary operation-symbols applied to a variable $\gamma : \Gamma$ as in: $\tau_n \left(\tau_{n-1} \left(\cdots \left(\tau_2 \left(\tau_1 \left(\gamma\right)\right)\right)\right)\right)$. Composable unary-operation symbols are ones which have compatible domains and codomains in the usual sense as in set theory.

We now propose a more precise version of the above claim: the terms of the sketch defined above form the elements of a free monoid over S. Before we can continue, we should decide what our term composition will be.

Definition 5 (Composition of terms)**.** Composition of terms is by substitution for the variable: for a term $\sigma : \Delta \vdash \Gamma$ and some terms τ_i with $\tau_n : \cdots \vdash \Xi$ we define

$$\left(\tau_n \left(\tau_{n-1} \left(\cdots \left(\tau_2 \left(\tau_1 \left(\gamma\right)\right)\right)\right)\right)\right) \circ \sigma = \tau_n \left(\tau_{n-1} \left(\cdots \left(\tau_2 \left(\tau_1 \left(\sigma \left(\delta\right)\right)\right)\right)\right)\right) : \Delta \vdash \Xi.$$

With our notion of composition in hand, we can now handwave an argument for our revised claim that the terms of the sketch form the elements of a free monoid. The reader will recall from our early discussions of basic type theory that substitution is associative [TODO in Ch 1]. As a consequence, any elementary sketch, including this one, satisfies that axiom for free. The identity term is given by zero composable unary operation-symbols applied to a variable. It's just a variable; when composing the identity term with any other term, we end up getting exactly that original term.

This loose argument is somewhat satisfying, but we can do better. To get there, we will first develop a notion generalizing *actions* from algebra. After doing so, we will give more concrete meaning to this sketch and complete our intuitive handle on it.

Definition 6. A **model** (also known as an algebra, an interpretation, a covariant action) of an elementary sketch is comprised of:

1. an assignment of a set A_X to each sort X and
2. an assignment of a function $\tau_* : A_X \to A_Y$ for each operation-symbol of the appropriate arity such that:
3. each law is preserved; i.e., for each law as before we have

$$\tau_{n_*}\left(\tau_{n-1_*}\left(\cdots\tau_{2_*}\left(\tau_{1_*}(x)\right)\cdots\right)\right) = \sigma_{m_*}\left(\sigma_{m-1_*}\left(\cdots\sigma_{2_*}\left(\sigma_{1_*}(x)\right)\cdots\right)\right)$$

that is, the covariant action on operation-symbols is faithful in the sense defined above.

The next definition will feature prominently in our later study of type theory, but will also prove immediately useful in studying Example 1 by forming the sets of a "for-free" model for any elementary sketch.

Definition 7. Given an elementary (unary) sketch, the **clone** at (Γ, X) is the set $\mathrm{Cn}_{\mathcal{L}}(\Gamma, X)$ of all the **terms** of sort X assuming a single variable of sort Γ, quotiented by the laws of the sketch.

The fact that a sketch's clones contain *equivalence classes* (with respect to the laws) of its terms will feature prominently in our later study of ideas central to the goals of this thesis. In particular, clones alone don't allow for any meaningful discussion of computational behavior of terms undergoing reduction; a term's normal form and its various reducible forms are identified in the clone.

It can be shown that the clones of a sketch form (the sets for) a model of a sketch. In particular, it can be shown that the sketch acts covariantly on the set of its clones:

Theorem 1. Every elementary sketch has a faithful covariant action on its clones $\mathcal{H}_X = \cup_\Gamma \mathrm{Cn}_{\mathcal{L}}(\Gamma, X)$ by sequencing with the operation symbol. Substitution for the (single) variable in a term gives a faithful contravariant action on $\mathcal{H}^Y = \cup_\Theta \mathrm{Cn}_{\mathcal{L}}(Y, \Theta)$.

Proof. The actions of $\tau : X \to Y$ on $\mathrm{Cn}_{\mathcal{L}}(\Gamma, X) \subseteq \mathcal{H}_X$ and $\mathrm{Cn}_{\mathcal{L}}(Y, \Theta) \subseteq \mathcal{H}^Y$ are given by:

- $\tau_* a_n\left(\cdots a_2\left(a_2\left(\sigma\right)\right)\cdots\right) = \tau\Big(a_n\Big(\cdots\big(a_2\left(a_1\left(\sigma\right)\right)\big)\Big)\Big) \in \mathrm{Cn}_{\mathcal{L}}\left(\Gamma, Y\right)$
- $\tau^*\zeta_m\left(\cdots\zeta_2\left(\zeta_1\left(y\right)\right)\right) = \zeta_m\Big(\cdots\zeta_2\big(\zeta_1\left(\tau(x)\right)\big)\Big) \in \mathrm{Cn}_{\mathcal{L}}\left(X, \Theta\right)$

where $\sigma : \Gamma, x : X,$ and $, y : Y$. Covariance of the former is clear. Contravariance of the latter follows by considering the behavior of substitutions in sequence. □

Theorem 2 (The canonical elementary language)**.** Every elementary sketch $\mathcal{L}$ presents a category $\mathrm{Cn}_{\mathcal{L}}$ via the for-free action on its clones, and conversely any small category C is presented by some sketch $\mathcal{L}$ in the sense that $C \cong \mathrm{Cn}_{\mathcal{L}}$. We write $\lceil - \rceil$ for this isomorphism and call the sketch $L\left(C\right) = \mathcal{L}$ the *canonical elementary language* of C.

The language is defined as follows:

- The sorts $\lceil X \rceil$ of $L\left(C\right)$ are the objects X of C
- The operation symbols $\lceil f \rceil$ are the morphisms f of C and
- The laws are $\lceil \mathrm{id} \rceil\left(x\right) = x$ and $\lceil g \rceil\left(\lceil f \rceil\left(x\right)\right) = \lceil g \circ f \rceil\left(x\right)$

and the isomorphism is clear.

Recalling our sketch of a monoid from Example 1, the substance of this covariant action morally amounts to saying that the sketch acts on its terms by left multiplication (here "multiplication" is actually just juxtaposition plus some parentheses) which gives the robust version of the handwavy argument we provided above.

Joke 1. A couple of type theorists walk into a Michelin starred restaurant. The menu reads in blackboard bold letters "ℕO UBTITUTIONS". They promptly leave.

1.1.3 The category of contexts and substitutions

We now introduce a very special category. This category is special in both the structure it enjoys as well as the central role it will play in the rest of the thesis. This category goes by many names: *syntactic category*, the (rather verbose) *category of contexts and substitutions*, and the elusive *classifying category*. We endeavor to explain the meaning behind each of these names over the course of the thesis, but for now we adopt the name most closely describing its presentation.

Definition 8 (The category of contexts and substitutions)**.** Given a sketch $\mathcal{L}$, the **category of contexts and substitutions**, written $\mathrm{Cn}^{\times}_{\mathcal{L}}$ is presented as follows:

- The objects are the contexts of $\mathcal{L}$, i.e., finite lists of distinct variables and their types.
- The generating morphisms are:
  - Single substitutions or *declarations* $[a/x] : \Gamma \to [\Gamma, x : X]$ for each term $\Gamma \vdash a : X$. The direction in the signature should be confusing unless you're either already an expert or a total novice to type theory.

– Single omissions or *drops* $\widehat{x} : [\Gamma, x : X] \to \Gamma$ for each variable $x : X$.

- The laws are given by an extended version of the familiar substitution lemma from type theory. The following laws are added for each collection of terms a, b and distinct variables x and y such that x does not appear free in a and y appears free in neither a or b:

$$\begin{aligned} [a/x]\,;\widehat{x} &= \mathrm{id} \\ [a/x]\,;[b/y] &= \left[[a/x]^*\, b/y\right];[a/x] \\ [a/x]\,;\widehat{y} &= \widehat{y};[a/x] \\ \widehat{x};\widehat{y} &= \widehat{y};\widehat{x} \\ [x/y]\,;\widehat{x};[y/x]\,;\widehat{y} &= \mathrm{id} \end{aligned}$$

We will briefly speak to the meaning of the laws. The first law says that binding a variable to some term and then forgetting the variable is just the same as doing nothing. The second law says that successive variable declarations commute *up to accounting for the first declaration in the body of the second*. The third law says that *non-overlapping* declarations and drops commute. The fourth law says that pairs of non-overlapping drops commute. The last law is tricky and is easier to explain by passing to the substitution point-of-view. Since the change of base functor is contravariant, this requires considering the compositions in reverse order as:

$$\widehat{y}^*;[y/x]^*\,;\widehat{x}^*;[x/y]^* = \mathrm{id}^*$$

Rendered thus, this law means that introducing a free variable y to the context[2], followed by replacing every free occurrence of x with y, followed by re-introducing x as a variable in the context, and then finally replacing every free occurrence of y with x is the same as doing nothing. More concisely at the expense of precision, renaming a free variable in a term and then un-renaming it results in the same term.

This category serves to allow us to define a special class of functor. In our case, that class of functor captures what it means to produce a model of an elementary sketch. The proof of this theorem is rather bureaucratic, but its importance is that it teaches us that the canonical elementary language of a category is purpose-built so that its models are precisely set-valued functors out of the category in question.

1.1.4 Models are essentially set-valued functors out of the category of a sketch

Theorem 3 (The classifying category)**.** Let $\mathcal{L}$ be an elementary sketch and $\mathrm{Cn}_{\mathcal{L}}$ the category it presents. Then the models of $\mathcal{L}$ correspond to functors $\mathrm{Cn}_{\mathcal{L}} \to \mathfrak{Set}$.

[2]possibly having no effect if y is already present

Proof. ($\Rightarrow$) Suppose we have an $\mathcal{L}$-model A. Then A is an assignment of a set $\lceil X \rceil_A$ to each sort $\lceil X \rceil$ and an assignment of a function $\lceil r \rceil_A : X_A \to Y_A$ to each operation-symbol $X \vdash \lceil r \rceil (x) : Y$ such that the laws (given by Theorem 2) of $\mathcal{L}$ are preserved. These assignment form precisely the data of a functor

$$\begin{aligned} F_A : \mathrm{Cn}_{\mathcal{L}} &\to \mathfrak{Set} \\ X &\mapsto \lceil X \rceil_A \\ \left(X \xrightarrow{r} Y \right) &\mapsto \lceil r \rceil_A \end{aligned}$$

.

It remains to show functoriality of these assignments which follow from the laws of the canonical elementary language and faithfulness of the model.

($\Leftarrow$) Suppose we have a functor $F_A : \mathrm{Cn}_{\mathcal{L}} \to \mathfrak{Set}$. We will construct a model A of $\mathcal{L}$ from F_A as follows: Recall from Theorem 2 that the sorts $\lceil X \rceil$ of the sketch $\mathcal{L}$ are precisely the objects X of $\mathrm{Cn}_{\mathcal{L}}$, and the operation symbols $X \vdash \lceil f \rceil : Y$ are the morphisms f of $\mathrm{Cn}_{\mathcal{L}}$. Now,

1. For each sort $\lceil X \rceil$ we assign $\lceil X \rceil_A = F_A (X)$.
2. For each operation symbol $\lceil f \rceil$ we assign $\lceil f \rceil_A = F_A (f)$.
3. Again by Theorem **??**, the only laws of the sketch are that $\lceil \mathrm{id} \rceil (x) = x$ and $\lceil g \rceil (\lceil f \rceil (x)) = \lceil g \circ f \rceil (x)$. According to the assignments in the previous two points, the first law says that $F_A (\mathrm{id}_X) = \mathrm{id}_{\lceil X \rceil_A}$, and the second says that $(F_A \lceil g \rceil) \circ (F_A (\lceil f \rceil)) = F_A (g \circ f)$. Both are ensured by functoriality.

□

1.2 Algebraic theories and their algebras

Having defined elementary sketches, which give us a way to define multi-sorted theories, it's obvious to request the ability to define multi-input operations[3]. Algebraic theories generalize the doctrine of elementary sketches and allow us to do so. As we upgrade our doctrine to allow products, many of the notions (*terms, clones, syntactic category, etc.*) which we developed in the simplified world of elementary sketches will come along for the ride.

Definition 9 (Algebraic theory)**.** A (finitary many-sorted) **algebraic theory** $\mathcal{L}$ has

1. a collection Σ of base types or **sorts** X

[3]Here's a little known statistic: At least one in two readers of this draft will observe that the doctrine of algebraic theory can be rephrased in terms of operards: algebraic theories are operads for which the tensor product used in forming the operation domains happens to be the plain ol' Cartesian product [**TODO: Nlab**]

2. an inexhaustible collection of variables $x_i : X$ of each sort;

3. a collection of **operation symbols**, $X_1, \ldots, X_k \vdash r : Y$ each having an **arity**, namely a list of input sorts X_i, and an output sort Y; and

4. a collection of **laws**, posed as equalities between different terms (in the sense defined before)

The next major concept we will introduce generalizes to algebraic theories the notion of *action* or *model* we saw previously for elementary sketches. As expected, the definition will be essentially the same up to taking some products. Before doing so, we will give an intervening example of an algebraic theory.

Example 2 (Algebraic theory of *ring*)**.** We sketch an algebraic theory encoding the familiar structure of a ring from abstract algebra. The presentation should look familiar (when squinting) to anyone with a background in abstract algebra, except that we force the existence of multiplicative and additive identities by requiring any model (to be defined!) of this theory to provide *global elements*, namely operations out of a distinguished sort $\mathbb{1}$.

1. Sorts: The sorts are $\mathbb{1}, S$. The variable collections for each sort are $\{\,\square\,\} \cup \{\,\square_i\,\}_i$ and $\{\,s_i\,\}_i \cup x, y, z$ respectively.

2. Operations:

$$\begin{aligned} \cdot &: S \times S \to S, \\ + &: S \times S \to S, \\ 0 &: \mathbb{1} \to S, \\ 1 &: \mathbb{1} \to S, \\ - &: S \to S \end{aligned}$$

3. Laws:

$$\begin{aligned} +(x, y) &= +(y, x) \\ +(0(\square), x) &= x \\ +(x, -(x)) &= 0(\square) \\ \cdot(x, y) &= \cdot(y, x) \\ \cdot(1(\square), x) &= x \\ \cdot(x, +(y, z)) &= +(\cdot(x, y), \cdot(x, z)) \end{aligned}$$

Having given the obligatory concrete example, we now have permission to proceed with another abstract definition: that of an *$\mathcal{L}$-algebra* for an algebraic theory:

Definition 10 ($\mathcal{L}$-algebra)**.** Given an algebraic theory $\mathcal{L}$ and a category C with finite products (in the sense of the universal property as treated in the chapter on basic category theory) an *$\mathcal{L}$-algebra in C* is comprised of

1. an object A_X of C for each sort X of $\mathcal{L}$, and
2. for each operation symbol $X_1, \ldots, X_k \vdash r : Y$, an assignment of a map $r_A : A_{X_1} \times \cdots \times A_{X_k} \to A_Y$ in C.

such that the assignments respect the laws of $\mathcal{L}$.

We are now in good shape to give an example of an algebra (in the category of sets) for the theory of a ring given in Example 2.

Example 3. 1. For the sorts, we set $A_S = \mathbb{Z}$ and $A_{\mathbb{1}} = \{\star\}$

2. For the operations, we set
   (a) $\cdot_A = *$ where $*$ is the ordinary multiplication of integers
   (b) $+_A = +$ where the second plus is ordinary addition of integers
   (c) 0_A to the constant function $x \mapsto 0 \in \mathbb{Z}$
   (d) 1_A to the constant function $x \mapsto 1 \in \mathbb{Z}$
   (e) $-_A$ to the function $x \mapsto -x$ taking an integer to its additive inverse
3. Verifying the rest of the laws is routine after understanding how to verify any of them, so we demonstrate just one. We show that the 0 selected by the model indeed serves as the left identity of addition in the model.

   Proof.

$$\begin{aligned} +_A \circ \langle 0_A, \mathrm{id} \rangle &= (x : \{\star\}, y : \mathbb{Z}) \mapsto 0_A(x) + \mathrm{id}(y) \\ &= (x : \{\star\}, y : \mathbb{Z}) \mapsto 0_{\mathbb{Z}} + y \\ &= (x : \{\star\}, y : \mathbb{Z}) \mapsto y \\ &= \mathrm{id}_{\mathbb{1}_{\mathbb{Z}} \times S_A} \end{aligned}$$

   Our proof is almost done, but we must justify that the final identity morphism is actually the interpretation of the variable (regarded as a term) x. This fact will be validated by results later in this section. In particular, Definition 12 will give a proper treatment to the interpretation of terms in an algebraic theory and allow us to justify the equivalence of the terms x and $\widehat{\square}^* x$ by[4] way of our construction of the syntactic category. With the clearing of that remaining goal-post deferred, we have shown what is required for this law. □

The reader familiar with algebra will observe that this example amounts to verifying that the integers form a ring under the standard multiplication and addition operations we learn in elementary school. A natural next question to ask is how we might encode a ring homomorphism in this framework. To answer this question, we define a more general notion:

[4] recall from Definition 8 that $\square$ is the variable we settled on for the sort $\mathbb{1}$ and substitution by the hat is context weakening or adding the variable

Definition 11 ($\mathcal{L}$-algebra homomorphism)**.** A *homomorphism* $A \to B$ of $\mathcal{L}$-algebras A and B in some category $\mathfrak{C}$ with finite-products is an assignment to each sort X of a $\mathfrak{C}$-morphism $\phi_X : A_X \to B_X$ between the corresponding objects in each algebra such that diagrams of the following form commute:

$$\begin{array}{ccc} A_{X_1} \times \cdots \times A_{X_k} & \xrightarrow{r_A} & A_{X_0} \\ \downarrow{\scriptstyle \phi_{X_0}} & & \downarrow{\scriptstyle \phi_{X_1} \times \cdots \times \phi_{X_k}} \\ B_{X_1} \times \cdots \times B_{X_k} & \xrightarrow{r_B} & B_{X_0} \end{array}$$

Remark 1. The following is an observation due to Lawvere in a paper written in his time teaching at Reed College [**lawvere_functorial_1963**]. The familiarity of this diagram is no mistake: indeed, by analogy to Theorem 3, we may understand algebras as product-preserving functors. A mapping between algebras then is a natural transformation, hence the naturality diagram in Definition 11. We will later make this analogy more concrete in Theorem 4.

Remark 2. Predictably, the $\mathfrak{C}$-valued algebras and homomorphisms of an algebraic theory $\mathcal{L}$ form a category, called $\mathcal{M}od_{\mathfrak{C}}(\mathcal{L})$.

Proof. Per Remark 1, we consider algebras and their homomorphisms as functors and natural transformations respectively. The identity morphisms are the identity natural transformations whose component morphisms are the identities of $\mathfrak{C}$. Composition of natural transformations is given by composition of their component morphisms, hence we may out-source the associativity condition to that guaranteed by the categorical structure of $\mathfrak{C}$. □

Most questions in type theory are concerned with the *terms* of the theory at hand. Normalization theorems talk about the accessibility (under some reduction relation) of a certain class of terms from any arbitrary term. Canonicity, a stronger property implying normalization, talks about the accessibility of another more strict class of terms from arbitrary start terms. These are but two examples of a broad spectrum of properties one might desire of the terms of a theory. Considering the primacy of term properties in type theory, it is rather strange that the notion of semantics we have built so far makes no commentary on terms besides the action on the clones given in Theorem 1. Our models so far have only given meaning to the individual *sorts* (types) and individual *operation symbols* (constructors) of the theory considered. In fact, this is enough: our models extend canonically to contexts and substitutions and thus give meaning to terms.

Definition 12 (Extending a model to terms)**.** Let A be an $\mathcal{L}$-algebra in a category $\mathfrak{C}$. This algebra extends canonically to an interpretation $[\![-]\!]$ of contexts by the following definition recursive in the structure of contexts:

$$[\![\emptyset]\!] = \mathbb{1}_C \tag{1.1}$$

$$[\![\Gamma, x : X]\!] = [\![\Gamma]\!] \times A_X \tag{1.2}$$

The (overly) careful reader will complain that $\mathfrak{C}$ doesn't necessarily feature a terminal object, but it turns out that a terminal object is guaranteed[5] by the finite product closure we imposed on $\mathfrak{C}$ in our definition of algebras. We are good to go.

Recalling more from the definition of an algebra, we know that A gives meaning to each operation symbol $Y_1, \dots, Y_k \vdash r : Z$ as a morphism $r_A : A_{Y_1} \times \cdots \times A_{Y_k} \to A_Z$ and gives meaning to each constant $c : Z$ by a morphism $1_{\mathfrak{C}} \to A_Z$. We can extend this uniquely to arbitrary terms in the context $\Gamma \equiv [\![x_1 : X_1, \dots, x_n : X_n]\!]$ by the following recursive definition:

$$[\![x_i]\!] : [\![\Gamma]\!] \equiv A_{X_1} \times \cdots \times A_{X_n} \xrightarrow{\pi_i} A_{X_i} \tag{1.3}$$

$$[\![c]\!] : [\![\Gamma]\!] \xrightarrow{<_!} \mathbb{1}_C \xrightarrow{c_A} A_Z \tag{1.4}$$

$$[\![r\,(u_1, \dots, u_k)]\!] : [\![\Gamma]\!] \xrightarrow{\langle [\![u_1]\!], \dots, [\![u_k]\!] \rangle} A_{Y_1} \times \cdots \times A_{Y_k} \xrightarrow{r_A} A_Z \tag{1.5}$$

where the $[\![u_i]\!]$ are the interpretations of the sub-expressions of the expression in the final line, π_i is the ith projection guaranteed to us by the universal property of products, and $<_!$ is the unique map into the terminal object. The angle bracket notion is used to express the product functor's action on morphisms in $\mathfrak{C}$. For clarity, we write out explicitly the composites for the reader:

$$\begin{aligned} [\![x_i]\!] &\equiv \pi_i \\ [\![c]\!] &\equiv c_A \circ <_! \\ [\![r\,(u_1, \dots, u_k)]\!] &\equiv r_A \circ \langle [\![u_1]\!], \dots, [\![u_k]\!] \rangle \end{aligned}$$

Theorem 4 (The classifying category of an algebraic theory)**.** Let $\mathcal{L}$ be an algebraic theory. Then

1. $\mathrm{Cn}^{\times}_{\mathcal{L}}$ has finite products and an $\mathcal{L}$-algebra.
2. Let $\mathfrak{C}$ be another category with a choice of finite products and an $\mathcal{L}$-algebra. Then the functor $[\![-]\!] : \mathrm{Cn}^{\times}_{\mathcal{L}} \to \mathfrak{C}$ preserves finite products and the $\mathcal{L}$-algebra, and is the unique such functor.
3. Any functor $\mathrm{Cn}^{\times}_{\mathcal{L}} \to C$ which preserves finite products also preserves the $\mathcal{L}$-algebra.

Proof.

1. We first show that the syntactic category has finite products. Recall that the objects of the syntactic category are variable contexts $[x : X, y : Y, z : Z, \dots]$. For any other context $[t : T, u : U, v : V, \dots]$ we have the product

   $$[x : X, y : Y, z : Z, \dots] \times [t : T, u : U, v : V] = [x : X, y : Y, z : Z, \dots, t : T, u : U, v : V, \dots]$$

   That is, products are given by concatenation of contexts. Now the model is given as follows:

[5] as the nullary finite product

(a) The sorts X of $\mathcal{L}$ are interpreted as single variable contexts $[x : X] \in \operatorname{ob} C$ where the variable x is arbitrary.

(b) The operation symbols $X_1, X_2, \cdots \vdash r : Y$ of $\mathcal{L}$ are interpreted as substitutions $\left[r\left(x_1, x_2, \ldots\right)/y\right] : [x_1 : X_1, x_2 : X_2, \ldots] \to [y : Y]$

2. The functor promised is precisely the one defined by Definition 12. TODO: we still need to verify the uniqueness of this functor, which Taylor himself never demonstrates.

3. This result demands a full proof, which isn't given in Taylor.

Proof. Suppose we have a functor $F : \mathrm{Cn}_{\mathcal{L}}^{\times} \to \mathfrak{C}$ which preserves products. We will show that it preserves the $\mathcal{L}$-model, in particular, that it takes the interpretation in $\mathrm{Cn}_{\mathcal{L}}^{\times}$ of any context Γ to the interpretation of Γ in $\mathfrak{C}$. TODO.

Note: Taylor claims the proof of this is given in [**lawvere_functorial_1963**], but I am (so far, I haven't tried too long) unable to identify the result in that paper, probably because Lawvere's formulation of algebraic theories is very different from Taylor's. I plan to tie up this loose end sometime soon. □

□

Chapter 2

Functorial semantics of the simply typed lambda calculus in cartesian closed categories

"Eeny, meeny, miny, moe"

Alonzo Church (Allegedly, on his choice of λ as the name for his calculus.)

This chapter exploits the heavy machinery developed in the previous chapter to give meaning, in specially structured categories, to the types and terms of the simply typed lambda calculus. In particular, we will present the lambda calculus as an *algebraic theory* and leverage our existing theory to give an interpretation of lambda terms in any *cartesian closed* category with an interpretation of the base types. We will begin by presenting the term language and type system for the calculus under consideration.

2.1 The simply typed lambda calculus

We'll work with the simply typed lambda calculus with a boolean base type. We will not dwell on the details of standard aspects of this development and refer the reader to the standard references ([**pierce_types_2002**], [**harper_practical_2016**]) for the full story.

Definition 13 (Types)**.** The types $\tilde{T}$ of the simply typed lambda calculus are generated by the following grammar:

$$\tau \quad \mathbb{1} \mid \tau_1 \to \tau_2 \mid \tau_1 * \tau_2$$

Definition 14 (Terms)**.** The terms of the simply typed lambda calculus (with booleans) are generated by the following grammar:

$$t \quad x \mid () \mid \pi_1 t \mid \pi_2 t \mid (t_1, t_2) \mid t_1 t_2 \mid \lambda x : \tau. t$$

where the variables x are drawn from a countably infinite set V.

The typing rules are given as follows:

Definition 15 (Typing rules)**.**

$$\frac{\Gamma \vdash t : \tau_1 * \tau_2}{\Gamma \vdash \pi_i t : \tau_i} \qquad \frac{\Gamma \vdash t_i : \tau_i}{(\tau_1, \tau_2) : \tau_1 * \tau_2}$$

$$\frac{\Gamma \vdash t_1 : \tau' \to \tau \qquad \Gamma \vdash t_2 : \tau'}{\Gamma \vdash t_1 t_2 : \tau} \qquad \frac{\Gamma, x : \tau' \vdash t : \tau}{\Gamma \vdash \lambda x : \tau'.\, t : \tau' \to \tau}$$

$$\frac{}{\Gamma \vdash x : \tau}\ (\text{x: }) \in$$

$$\frac{}{\Gamma \vdash () : \mathbb{1}}$$

2.2 Lambda theories, beta-eta rules, and cartesian closed categories

We now define a notion which mediates between full-cartesian closed structure and having lambda abstraction at all. Our definition comes from Taylor [**taylor_practical_1999**].

Definition 16 (Raw cartesian closed structure)**.** Let $\mathfrak{C}$ be a category with a specified terminal object $\mathbb{1}$ and specific products together product projections. We say that $\mathfrak{C}$ is *raw cartesian closed* or *has raw cartesian closed structure* if we have the following for each pair of objects X, Y of $\mathfrak{C}$:

1. An object Y^X
2. An morphism, *application*, $\mathrm{ev}_{X,Y} : Y^X \times X \to Y$ and
3. For each object Γ, a function of hom-sets $\lambda_{\Gamma,X,Y} : \mathfrak{C}(\Gamma \times X, Y) \to \mathfrak{C}\left(\Gamma, Y^X\right)$ obeying the naturality law:

$$\lambda_{\Gamma,X,Y}(\mathfrak{p} \circ (\mathfrak{u} \times \mathrm{id}_X)) = \lambda_{\Gamma,X,Y}(\mathfrak{p}) \circ \mathfrak{u}$$

for each $\mathfrak{u} : \Gamma \to \Delta$ and $\mathfrak{p} : \Delta \times X \to Y$.

Most sources define the following only by *context clues*, which is a technical term meaning that they don't define it at all. We give an explicit characterization of *lambda theories*:

Definition 17 (Lambda theory)**.** An algebraic theory $\mathcal{L}$ is a *lambda theory* if its classifying category $\mathrm{Cn}^{\times}_{\mathcal{L}}$ has raw cartesian closed structure.

Definition 18 (Beta-eta rules)**.** A lambda-theory $\mathcal{L}$ satisfies the $\beta - \eta$ *rules* if for all $\mathfrak{p} \in \mathrm{Cn}_{\mathcal{L}}(\Gamma \times X, Y)$ we have

$$\mathrm{ev}_{X,Y} \circ \left(\lambda_{\Gamma,X,Y}(\mathfrak{p}) \times \mathrm{id}_X\right) = \mathfrak{p} \qquad (\beta)$$

$$\lambda_{Y^X,X,Y}\left(\mathrm{ev}_{X,Y}\right) = \mathrm{id}_{Y^X} \qquad (\eta)$$

These deserve some commentary. The beta rule says that application of an abstracted body $\mathfrak{p}$ to the identity gives you back the body you started with: the beta rule forces that application of lambda expressions does nothing more than substitute for the abstracted variable. TODO: explain eta.

We now define a notion more well-known outside of computer science. Cartesian closed structure endows a category with the ability to take products and function spaces over its objects in a suitable fashion. It will turn out that this familiar form is equivalent to the combination of raw cartesian closed structure and beta-eta rules.

Definition 19 (Cartesian closed structure)**.** A category $\mathfrak{C}$ is *cartesian closed* if it has all products and exponentials.

The category $\mathfrak{Set}$ is the prototypical cartesian closed category whose products are the usual cartesian products of sets and whose exponentials are function spaces: for sets X and Y, $Y^X = \{\text{functions } f \mid \mathrm{dom}(f) = X \wedge \mathrm{cod}(f) = Y\}$. Another good source of example cartesian closed categories is topos theory. All elementary toposes, including categories of presheaves, are cartesian closed ([**leinster_informal_2011**].) It turns out that all of these examples of cartesian closed categories can also be characterized as the classifying categories of lambda theories satisfying the beta-eta rules.

Theorem 5 (Finite-product category + raw CCS + beta-eta $\iff$ cartesian closed)**.** Let $\mathfrak{C}$ be a category. Then $\mathfrak{C}$ is cartesian closed if and only if $\mathfrak{C}$ has finite products, a specified terminal object, and both is raw cartesian closed and satisfies the beta-eta laws.

Proof. TODO □

2.2.1 An algebraic theory of the lambda calculus

Finally, we will present the lambda calculus as a series of algebraic theories, each of which are identical except for the equations they admit. These theories will correspond to different instances of the lambda calculus with more or less terms identified according to the usual $\beta\eta\alpha$ conversion rules.

Definition 20 (Algebraic theory of the lambda calculus)**.** The α*-lambda calculus* is presented as the following algebraic theory:

1. Sorts: We define the sorts Σ_λ to be the set $\tilde{T}$ given in Definition 13.
2. Variables: The variables are drawn from some countably infinite set V as in the definition of terms.

3. We define the following operation symbols for each $\tau_1, \tau_2, \tau, \tau' \in \Sigma_\lambda$:

$$t : \tau_1 * \tau_2 \vdash \pi_1 (t) : \tau_1$$
$$t : \tau_1 * \tau_2 \vdash \pi_2 (t) : \tau_2$$
$$[t_1 : \tau_1, t_2 : \tau_2] \vdash (t_1, t_2) : \tau_1 * \tau_2$$

$$[t : \tau' \to \tau, t' : \tau'] \vdash t\, t' : \tau$$

$$\vdash () : \mathbb{1}$$

   The reader will note that we have not yet added an operation symbol for lambda abstraction. This requires careful consideration, because lambda abstraction is not really an operation symbol but rather a family of operation symbols defined mutually with the others. For each term $\Gamma, x : \tau' \vdash t : \tau$, we add a new operation symbol

$$\Gamma \vdash (\lambda (x : \tau') . t) (\vec{\gamma}) : \tau \quad \text{(where } \vec{\gamma} \text{ is the list of variables comprising } \Gamma)$$

   The operation symbols are the least (TODO: does this even make sense) set closed under taking abstractions in this way.

4. The equations are only those of the α-rule which says that terms can be identified up to renamings of their bound variables. We notably do not include the $\beta\eta$-rules in this version.

$$\lambda (x : X) . t = \lambda (x' : X) . \left[x'/x\right]^* t \qquad (\alpha)$$

   defined for each operation symbol $f : Y^X$ and terms $\Gamma, x : X \vdash t : \tau$ and $\Gamma \vdash t' : X$.

Theorem 6 (α-lambda calculus is raw cartesian closed)**.** The α-lambda calculus is raw cartesian closed

Proof. TODO see Taylor □

Definition 21 (Lambda calculus)**.** The lambda calculus, or the $\alpha\beta\eta$-lambda calculus is defined as the algebraic theory of the α-lambda calculus with the following additional equational laws:

$$(\lambda (x : X) . t)\ t' = \left[t'/x\right] * t \qquad (\beta)$$
$$f = (\lambda (x : X) . f\, x) \qquad (\eta)$$

Theorem 7 (Lambda calculus is a lambda theory with beta-eta)**.**

Corrolary 1 (The classifying category of the lambda calculus is cartesian closed)**.** Immediate by the previous result and Theorem 5.

Corrolary 2 (λ-algebras are diagrams in cartesian closed categories)**.** Immediate by the previous result and Theorem 4.

Corrolary 3 (An interpretation of base types in a CCC gives rise to an interpretation of λ-terms)**.**

This final result will prove important in 4

Chapter 3

Normalization by evaluation: the classical perspective

In this chapter, we outline the technique of *normalization by evaluation* which exploits an existing programming language evaluator to produce a normalization function for the *open terms* of some other language. Our intent is merely to communicate the ideas involved with a view to drawing analogies to normalization by evaluation in our later discussion of gluing. In light of this, we describe the technique in full detail but omit all proofs of correctness (in the literature called "soundness") The chapter will follow our OCaml implementation of normalization by evaluation for Gödel's System T. Our implementation was inspired by the habilitation of Andreas Abel and the Wikipedia page on normalization by evaluation written by a number of authors.

```
open Core

type ty = Arrow of ty * ty
        | Prod of ty * ty
        | Nat [@@deriving sexp]

type var = string [@@deriving sexp]

let cur : int ref = ref 0

let fresh () : var = let res = Printf.sprintf "var%d" !cur in
                     cur := !cur + 1;
                     res

type syn = Lam of var * ty * syn
         | App of syn * syn

         | Pair of syn * syn
         | P1 of syn
         | P2 of syn
```

```
         | Zero
         | Succ of syn
         | Rec of ty * syn * syn * syn (* rec : forall (T: ty), T -> (N -> T -> T) -> N

         | Var of var [@@deriving sexp]

(* Note: There are no elimination forms in the semantics; they are represented
   as syntax through the SYN embedding. *)
type sem = LAM of (sem -> sem)
         | PAIR of sem * sem
         | SYN of syn
[@@deriving sexp]

(* reflect is a type-indexed family of functions taking the syntax into the
   semantics *)
let rec reflect (tau : ty) : syn -> sem =
  fun t ->
  match tau with
  | Arrow (a, b) ->
    LAM (fun s -> reflect b (App (t, (reify a s)) ))
  | Prod (a, b) ->
    PAIR ((reflect a (P1 t), (reflect b (P1 t))))
  | Nat ->
    SYN t

and reify (tau : ty) : sem -> syn =
  fun t ->
  match (t, tau) with
  | (LAM s, Arrow (a, b)) ->
    let x = fresh () in
    Lam (x, tau, reify b (s (reflect a (Var x))))
  | (PAIR (l, r), Prod(a,b)) ->
    Pair (reify a l, reify b r)
  | (SYN b, _) ->
    b
  | _ ->
    raise (Failure (Printf.sprintf "Reification failed on (%s : %s)"
                      ([%sexp_of : sem] t |> Sexp.to_string)
                      ([%sexp_of: ty] tau |> Sexp.to_string)))


let rec metarec (tau : ty) (z : 'a) (s : syn -> 'a -> 'a) : syn -> 'a =
  fun n ->
  match n with
```

```
  | Zero -> z
  | Succ n ->
    s n (metarec tau z s n)
  | u ->
    let v_z = reify tau z in
    let v_s = reify (Arrow(Nat, (Arrow (tau, tau)))) (LAM (fun n -> match n with
                                                                  | SYN n -> LAM(s n)
                                                                  | _ -> raise (Failure
    reflect tau (Rec (tau, v_z, v_s, u))

type context = (var * ty) list

(* A valuation is a list semantic terms to be interpreted as an assignment of
   terms to variables in an ordered context *)
type valuation = (var * sem) list


(* we assume that a bad lookup is never performed; this will end up being the
   case as long as we only take the meaning in the valuation generated by the
   context closing a well-typed term. *)
let lookup (x : var) (rho : valuation) : sem =
  List.rev (List.filter ~f:(fun entry -> String.equal (fst entry) x) rho)
  |> fun l -> List.nth_exn l 0
  |> snd

(* Given a context gamma, produces a valuation of reflected variables *)
let reflect_ctx (gamma : context) : valuation =
  List.map ~f:(fun (x,tau) -> (x, reflect tau (Var x))) gamma

(* interprets into the semantics an open term t with respect to a valuation rho
   which closes it *)
let rec interp (t : syn) (rho : valuation): sem =
  match t with
  | Var x ->
    lookup x rho
  | Lam(x, _, b) ->
    LAM (fun s -> interp b (List.append rho [(x,s)]))
  | App(e1, e2) ->
    let LAM e1f = interp e1 rho in
    e1f (interp e2 rho)
  | Pair(e1, e2) ->
    PAIR(interp e1 rho, interp e2 rho)
  | P1 e ->

    let PAIR (l,_) = interp e rho in
```

```
    l
  | P2 e ->
    let PAIR (_,r) = interp e rho in
    r
  | Zero -> SYN Zero
  | Succ n ->
    SYN (Succ n)
  | Rec(tau, z, s, n) ->
    let LAM si = interp s rho in (* s : nat -> A -> A *)
    let zi = interp z rho in
    let SYN ni = interp n rho in
    metarec tau zi (fun m -> let LAM(f) = si (SYN m) in f) ni
    [@@warning "-8"]
(* the above incomplete matches are irrefutable on well-typed terms t *)

let nbe (gamma : context) (tau : ty) : syn -> syn =
  fun t -> interp t (reflect_ctx gamma) |> reify tau

(* EXAMPLES:
    nbe [("y", Nat); ("z", Nat)] Nat (App (Lam("x", Nat, Var("y")), Var ("z")));;
    - : syn = Var "y"

   nbe [] Nat (App ((Lam ("z", Nat, (Rec(Nat, Zero, Lam("x",Nat,Lam("y",Nat,(Var "x"))),
   - : syn = Zero*)
```

Chapter 4

Normalization by gluing

> You, you were like glue
> holding each of us together
> I slept through July
> while you made lines in the heather
>
> Fleet Foxes, "Lorelai"

This chapter will consider the normalization problem for the simply typed lambda calculus using all of the magic technology we've developed so far. Normalization comes in both weak and strong flavors, but both are properties of a formal system together with a *reduction* relation, say, $- \to =$. By a *normal form* we mean a term t for which there is no other t' such that $t \to t'$. Writing $- \to^* =$ for the least transitive, reflexive closure of this relation, we say the reduction system enjoys *weak normalization* if for any well-typed term e, there exists a normal form n such that $e \to^* n$. A reduction system enjoying *Strong normalization* is one for which *every* reduction sequence ends in a normal term. *Metatheorems* of this kind tend to resist standard techniques like straight-forward induction over the types of the lambda calculus. Instead, metatheoreticians resort to the *occult* technique of logical relations. The method of logical relations is, broadly speaking, opaque even to experienced practitioners. The construction is easy to use in simple settings like that of the simply-typed lambda calculus (see [**pierce_types_2002**]) but becomes devilishly complicated when working with more complex lambda calculi.

The method of *Artin gluing*, torn from the topos theorist's cookbook, is a more transparent encoding of the mechanics of logical relations. Where the method of logical relations is ad-hoc and mysterious, gluing regularizes the thought involved and makes the choices involved in proofs by logical relations completely automatic. The *scone* or *Sierpinski cone* is a famous instance of the gluing construction used in and outside of type theory. The *scone* is not suited to our particular problem of *(open) normalization* however, as the scone is generally concerned with properties of *closed terms* with no free variables, whereas our construction will apply to terms in any context.

4.1 Variable-arity Kripke relations

TODO

4.2 The comma construction and friends

This section introduces a general construction that will, among other things, allow us to define the gluing construction. As hinted at before, gluing is just a special case of the comma. The idea of the comma is to glob some extra data to the objects of a category, and has a notion of an object-with-extra-data-morphism which preserves the attached data. This intuition perhaps underlies the name for the gluing construction which is a special case of the comma.

Definition 22 (Comma category)**.** Let E, D, C be categories and $F : D \to C \leftarrow E : G$ be a pair of functors sharing their codomain C. The comma category $F \downarrow G$ has as its

1. Objects: triples $\left(d : D, Fd \xrightarrow{f} Ge, e : E\right)$
2. Morphisms: arrows $(h, k) : (d, f, e) \to (d', f', e')$ are pairs of D and E-morphisms $h : d \to d', k : e \to e'$ making the following diagram commute:

$$\begin{array}{ccc} Fd & \xrightarrow{Fh} & Fd' \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle f'} \\ Ge & \xrightarrow[Gk]{} & Ge' \end{array}$$

4.2.1 An easy comma: the category of renamings

The category of renamings is something like a less proof-relevant version of the category of contexts and substitutions in the following sense: if there exists a substitution relating two contexts in syntactic category, then there exists a renaming relating the same contexts. In this sense, renamings allow us to track changes in context *due to a substitution* without actually using substitutions which, because substitutions are identified up to computation, are problematic in a development requiring distinguishing between terms at various stages of reduction (e.g. distinguishing between a term and its normal form.) The category of renamings allows us to define the presheaves we need while not losing track of how contexts evolve by substitution. These two points are precisely the desiderata satisfied by the category of renamings. We define the category and at the same time explain what its morphisms are in terms of the more familiar notion of substitution:

The category of renamings has a nice presentation as a comma category.

Definition 23 (Category of renamings)**.** Recall V the countably infinite set of variables and $\tilde{T}$ the set of all types in the simply typed lambda calculus from our early

definitions in Chapter 2. Let $\mathbb{F}$ be the category whose objects are finite subsets of V and whose morphisms are all functions between the underlying sets. Let $\mathbb{T} : \mathbb{F} \to \mathfrak{Set}$ be the functor which is constantly $\tilde{T}$ on objects and constantly $\mathrm{id}_{\tilde{T}}$ on morphisms. Let $U : \mathbb{F} \to \mathfrak{Set}$ be the forgetful functor taking finite variable sets to their underlying sets and functions to functions. Now by the *category of renamings* we mean the *opposite category* of the comma $U \downarrow \mathbb{T}$ whose

- Objects are *pairs* $\left(V : \mathbb{F}, \Gamma : V \to \tilde{T}\right)$, i.e., finite variable list together with an assignments of types to a each variable. Note that we elide the final entry of the triple of the comma objects, since it adds no extra information. These objects are precisely the contexts of the familiar classifying category.

- Morphisms of contexts $(V, \Gamma) \to (V', \Gamma')$ are functions $\rho : V \to V'$ making the following diagram commute:

$$\begin{array}{ccc} V & \xrightarrow{\rho} & V' \\ {\scriptstyle\Gamma}\downarrow & & \downarrow{\scriptstyle\Gamma'} \\ \tilde{T} & \xrightarrow[\mathrm{id}_{\tilde{T}}]{} & \tilde{T} \end{array}$$

  The reason the comma morphisms are single functions and not pairs is that the second function would not communicate any data, it would just be taken by $\mathbb{T}$ to the identity no matter what it is. We now unpack this means. The functions ρ are *type-preserving renamings*: for each variable $x \in V$, we have that $\Gamma'(\rho x) = \Gamma(x)$. In particular, this restricts ρ to the following classes of morphisms in the classifying category: substitutions of variables for variables of the same type, context extension with new variables, and all compositions of these.

We write $\mathrm{Ren}_\Sigma = (U \downarrow \mathbb{T})^{\mathrm{op}}$ for the category of renamings.

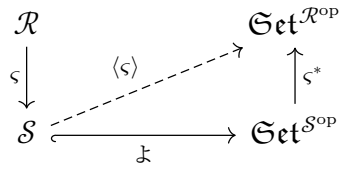
Remark 3. There is an obvious inclusion $\iota : \mathrm{Ren}_\Sigma \to \mathrm{Cn}^{\Box}_{\mathcal{L}}$ which takes contexts to contexts and casts renamings as change-of-variable substitutions. When going between Ren_Σ and $\mathrm{Cn}^{\Box}_{\mathcal{L}}$, we will take liberty to implicitly insert this coercion as needed without further warning. It turns out that this inclusion is *faithful* and thus witnesses Ren_Σ as a *subcategory* of the syntactic category.

4.2.2 A harder example: the gluing category; or, the category of proof-relevant Ren-Kripke relations over types

The gluing category is an instance of the comma construction which will star as the primary semantic domain in our development. To define it, we first need to define a functor which, in a loose sense that will be explained in due course, defines Ren_Σ-presheaves of *open* terms.

The relative hom functor

Every functor $\varsigma : \mathcal{R} \to \mathcal{S}$ induces the following situation:



That is, we get a functor

$$\langle\varsigma\rangle : \mathcal{S} \to \mathfrak{Set}^{\mathcal{R}^{\mathrm{op}}}$$

by adjusting the Yoneda embedding into the category of $\mathcal{S}$-presheaves. In particular, specializing to the case of the inclusion $\iota : \mathrm{Ren}_\Sigma \to \mathrm{Cn}^\square_\mathcal{L}$ of Remark 3 gives us a functor

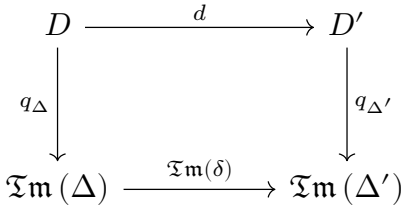
$$\begin{aligned}\mathfrak{Tm} : \mathrm{Cn}^\square_\mathcal{L} &\to \widehat{\mathrm{Ren}_\Sigma} \\ \Delta &\mapsto \mathrm{Cn}^\square_\mathcal{L}(\iota(-), \Delta)\end{aligned}$$

which can be construed as taking a syntactic context to a presheaf of open terms. This is a confusing idea at first, and it helps to consider the simple cases to understand it. Consider any singleton context $\tau : \mathrm{Cn}^\square_\mathcal{L}$. Then $\mathfrak{Tm}$ takes τ to the presheaf $\mathrm{Cn}^\square_\mathcal{L}(-, \tau)$ which takes any renaming context Γ to $\mathrm{Cn}^\square_\mathcal{L}(\Gamma, \tau)$. Recalling the definition of the syntactic category and its generating morphisms, the latter set comprises the terms of type τ closed under Γ, represented as single substitutions $[t/x] : \Gamma \to \tau$. Generalizing to multivariable target contexts Δ gives presheaves of *lists* of open terms of the types in Δ.

Gluing syntax to semantics along the relative hom functor

We at last define the *gluing category* as the comma of the category of renamings along the relative hom functor just defined.

Definition 24 (The gluing category)**.** The gluing category $\mathrm{Gl}_\mathcal{L}$ is defined as the comma $\widehat{\mathrm{Ren}_\Sigma} \downarrow \mathfrak{Tm}$. Explicitly, its objects are triples $D : \widehat{\mathrm{Ren}_\Sigma}, q : D \Rightarrow \mathfrak{Tm}(\Delta), \Delta : \mathrm{Cn}^\square_\mathcal{L}$. The objects of the gluing category are *proof relevant logical predicates*, in a sense that will be more clear after the next subsection. Following the definition of the comma construction, the morphisms $(D, q_\Delta, \Delta) \to (D', q_{\Delta'}, \Delta')$ are pairs $\left(d : D \to D', \delta : \mathrm{Cn}^\square_\mathcal{L}[\Delta', \Delta]\right)$ are pairs of a $\widehat{\mathrm{Ren}_\Sigma}$ natural transformation and a substitution making the following diagram commute:



We will better understand what all this means after understanding how the gluing category subsumes the notion of variable-arity Kripke relations.

The subcategory of (ordinary) variable-arity Kripke relations

We can recover ordinary, proof-*irrelevant* variable-arity Kripke relations as a subcategory of the gluing category. In particular, ordinary variable-arity Kripke relations arise as those objects $\left(D : \widehat{\mathrm{Ren}_\Sigma}, q : D \Longrightarrow \mathfrak{Tm}\left(\Delta\right), \Delta : \mathrm{Cn}_{\mathcal{L}}^{\square}\right)$ of the gluing category whose quotient map q is a component-wise monomorphism; i.e., for each $\Gamma \in \mathrm{Ren}_\Sigma$, we have that $q_\Gamma : D\left(\Gamma\right) \rightarrowtail \mathfrak{Tm}\left(\Delta\right)\left(\Gamma\right)$ is a monomorphism. Here's why. Recalling the definition of Ren_Σ-Kripke relations over $\tau \in \mathrm{Cn}_{\mathcal{L}}^{\square}$, we need to find in these data a Ren_Σ-indexed family $\left\{R_\Gamma\right\}_{\Gamma \in \mathrm{Ren}_\Sigma}$ of sets of $\mathrm{Cn}_{\mathcal{L}}^{\square}$-morphisms into τ subject to the following relative monotonicity condition: for any renaming $\rho : \Gamma' \to \Gamma$, if $t : \Gamma \to \tau$ is in R_Γ, then we have that $t \circ \rho : \Gamma' \to \tau$ is in $R_{\Gamma'}$. Looking again to our proposed Kripke predicate objects of the gluing category, we can see that the presheaf R together with the natural mono q define for each Γ a subset[1] of the substitutions $\Gamma \to \tau$. It is tempting to dismiss the naturality of q as bureaucracy; but naturality turns out to be essential to recovering the relative monotonicity condition. To see why this recovers what we had before, let us look at the naturality diagram of q:

$$\begin{array}{ccc} R\left(\Gamma\right) & \xrightarrow{\quad q_\Gamma \quad} & \mathfrak{Tm}\left(\Delta\right)\left(\Gamma\right) \\ \Big\downarrow{\scriptstyle R(\rho)} & & \Big\downarrow{\scriptstyle \mathfrak{Tm}(\Delta)(\rho)} \\ R\left(\Gamma'\right) & \xrightarrow{\quad q_{\Gamma'} \quad} & \mathfrak{Tm}\left(\Delta\right)\left(\Gamma'\right) \end{array} \qquad \begin{array}{c} t \\ \Big\downarrow \\ t \circ \rho \end{array}$$

As mentioned before, because they are monos we may identify the component morphisms with their images. We write $\left|q_\Gamma\right| \subseteq \mathfrak{Tm}\left(\Delta\right)\left(\Gamma\right)$ for the image, for each Γ. Now what this diagram says is that for any $q_\Gamma\left(r \in R\left(\Gamma\right)\right) = t \in \left|q_\Gamma\right|$, we have that $t \circ \rho = q_{\Gamma'}\left(R\left(\rho\right)\left(r\right)\right) \in \left|q_{\Gamma'}\right|$. This is precisely the relative monotonicity condition for $\mathfrak{C}$-Kripke relations: containment in each predicate is functorial with respect to renaming *up to renaming*. It turns out that the category of Ren_Σ-Kripke relations is a full subcategory of the gluing category $\widehat{\mathrm{Ren}_\Sigma} \downarrow \mathfrak{Tm}$.

Theorem 8 (Kripke full subcategory of gluing)**.** TODO: maybe, I don't really think I will use this result.

Proof. TODO □

The perspective gained above allows for a more conceptual understanding of the objects in the gluing category: the presheaves R define context-indexed families of *witnesses* to the inclusion of terms in the predicate, and q is a quotient map that forgets the difference between distinct witnesses $w, w' \in R\left(\Gamma\right)$ and whose image just

[1] this result is elementary for $\mathfrak{Set}$, but also holds for any *topos* modulo its notion of subobjects ([**leinster_informal_2011**])

records those terms $t \in \mathfrak{Tm}(\Delta)(\Gamma)$ taken to be in the predicate; insofar as q is a mono, these objects are exactly the variable-arity Kripke relations we considered classically at the beginning of this chapter.

TODO: explain intuition for the morphisms of the gluing category

4.3 Stratified syntax, theories, presheaves, algebras

In this section, we will enlist a motely crew of *presheaves of syntax* which more-or-less represent families of terms in the lambda calculus. These presheaves are defined over Ren_Σ for reasons that will become apparent later. We will ultimately define lambda algebras over these presheaves, for which we require some understanding of how to represent variables in $\widehat{\mathrm{Ren}_\Sigma}$ and how exponentiation in $\widehat{\mathrm{Ren}_\Sigma}$ corresponds to lambda abstraction. TODO: more discussion: why presheaves of syntax? explain why we even want algebras in the first place — it is in order to glue them to the semantic algebra on $\mathfrak{Tm}$

4.3.1 Variables and binding in Renhat

Definition 25 (Variable presheaf)**.** We can define a presheaf of *typed variables* in $\widehat{\mathrm{Ren}_\Sigma}$ with the Yoneda embedding on Ren_Σ:

$$\mathfrak{V}_\tau = \text{ﻉ}\tau = \mathrm{Ren}_\Sigma(-, \tau)$$

With that definition, we have $\mathfrak{V}_\tau(\Gamma) = \mathrm{Ren}_\Sigma(\Gamma, \tau)$ where the right-hand side is comprised of renamings $\Gamma \to [x : \tau]$ which are *functions* $\rho : \mathrm{dom}([x : \tau]) \to \mathrm{dom}(\Gamma)$ such that $\Gamma(\rho(x)) = [x : \tau](x) = \tau$. That is, the ρ are functions selecting a variable of type τ in Γ. More concisely, we have an isomorphism $\mathfrak{V}_\tau(\Gamma) \cong \{x \mid (x : \tau) \in \Gamma\}$ by which we allow ourselves to consider this a *presheaf of syntax*.

We can also explicate binding structure in $\widehat{\mathrm{Ren}_\Sigma}$ thanks to the Yoneda lemma. Let $\mathfrak{P} : \widehat{\mathrm{Ren}_\Sigma}$. Exponentiation by a representable/variable presheaf, $\mathfrak{V}_\tau = \text{ﻉ}\tau$ gives

$$\begin{aligned}
\mathfrak{P}^{\mathfrak{V}_\Delta}(\Gamma) &= \mathfrak{P}^{\text{ﻉ}\Delta}(\Gamma) \\
&= \widehat{\mathrm{Ren}_\Sigma}\left[\text{ﻉ}\Gamma \times \text{ﻉ}\Delta, \mathfrak{P}\right] && (2) \\
&\cong \widehat{\mathrm{Ren}_\Sigma}\left[\text{ﻉ}(\Gamma \times \Delta)\right] && \text{(since the Yoneda embedding is cartesian closed)} \\
&\cong \mathfrak{P}(\Gamma \times \Delta) && \text{(by the Yoneda lemma)}
\end{aligned}$$

Where step (2) follows from the definition of the exponentials in the category of presheaves, c.f. page 46 of [**mac_lane_sheaves_1992**].

The presheaves of typed variables together with the simplified view of exponentials in $\widehat{\mathrm{Ren}_\Sigma}$ allow us to define algebras for a new lambda theory NN of stratified *neutrals and normals* defined in the previous section, which allow for a more fine-grained discussion of normal terms.

4.3.2 Stratifying neutral and normal terms

We introduce a new type system over the syntax and type structure of the lambda calculus as defined in Chapter-2. This new type system will distinguish between *neutral terms*, a special class of *normal terms* which are in some sense beta-redexes whose reduction is *blocked* by a variable in the head term, and all other normal terms. In some sense, *neutral terms* can be regarded as those terms which are normal only because they're "stuck."

Definition 26 (Neutral and normal judgments)**.**

$$\frac{}{\Gamma \vdash_{\text{ne}} x : \tau}\ (x : \tau) \in \Gamma$$

$$\frac{\Gamma \vdash_{\text{ne}} M : \tau_1 * \tau_2}{\Gamma \vdash_{\text{ne}} \pi_i M : \tau_i} \qquad \frac{\Gamma \vdash_{\text{ne}} t_1 : \tau' \to \tau \qquad \Gamma \vdash_{\text{nf}} t_2 : \tau'}{\Gamma \vdash_{\text{ne}} t_1 t_2 : \tau}$$

$$\Gamma \vdash_{\text{nf}} () : \mathbb{1} \qquad \frac{\Gamma \vdash_{\text{nf}} N_i : \tau_i}{\Gamma \vdash_{\text{nf}} (N_1, N_2) : \tau_1 * \tau_2} \qquad \frac{\Gamma, x : \tau \vdash_{\text{nf}} b : \tau'}{\Gamma \vdash_{\text{nf}} \lambda x : \tau.\, b : \tau \to \tau'}$$

$$\frac{\Gamma \vdash_{\text{ne}} t : \theta}{\Gamma \vdash_{\text{nf}} t : \theta}\ (\theta \in T \text{ A BASE TYPE})$$

Definition 27 (Algebraic theory of neutrals and normals)**.** The judgments above give rise to a lambda theory with a richer sort structure than that of the lambda theory defined back in Chapter 2, because we differentiate between neutral terms of type τ and normal terms of type τ.

We define the lambda theory NN as the algebraic theory (c.f. Theorem 2) corresponding to the category with the following objects and generating morphisms:

1. The objects are the collection generated by taking products and exponentials over the collection $\Sigma = \left\{\mathcal{M}_\tau \mid \tau \in \tilde{T}\right\} \cup \left\{\mathcal{N}_\tau \mid \tau \in \tilde{T}\right\} \cup \left\{\mathcal{V}_\tau \mid \tau \in \tilde{T}\right\}$

2. Generating morphisms, for each $\tau, \tau' \in \tilde{T}$:

$$\begin{aligned}
\text{var}_\tau &: \mathcal{V}_\tau \to \mathcal{M}_\tau \\
\text{fst}_\tau^{\tau'} &: \mathcal{M}_{\tau\times\tau'} \to \mathcal{M}_\tau \\
\text{snd}_\tau^{\tau'} &: \mathcal{M}_{\tau'\times\tau} \to \mathcal{M}_\tau \\
\text{app}_\tau^{\tau'} &: \mathcal{M}_{\tau'\to\tau} \times \mathcal{N}_{\tau'} \to \mathcal{M}_\tau \\
\text{incl}_\theta &: M_\theta \to N_\theta \\
\text{unit} &: \mathbb{1} \to \mathcal{N}_{\mathbb{1}} \\
\text{pair}_\tau^{\tau'} &: \mathcal{N}_\tau \to \mathcal{N}_{\tau'} \to \mathcal{N}_{\tau\times\tau'} \\
\text{abs}_{\tau\to\tau'} &: {\mathcal{N}_{\tau'}}^{\mathcal{V}_\tau} \to \mathcal{N}_{\tau\to\tau'}
\end{aligned}$$

3. The laws are the usual $\alpha-$, $\beta-$, and $\eta-$rules.

Note that the domain of the unit operation symbol is merely the nullary product of sorts.

Remark 4. Any algebra $\{(\mathfrak{V}_\tau, \mathfrak{X}_\tau)\}_{\tau\in\widetilde{T}}$ of the lambda theory $\mathcal{L}_{\alpha\beta\eta}$ (TODO: make this terminology consistent) from Chapter 2 gives rise to an algebra $\mathfrak{A}$ of the theory NN by setting

$$\begin{aligned}\mathfrak{A}_{\mathcal{M}_\tau} &= \mathfrak{X}_\tau\\ \mathfrak{A}_{\mathcal{N}_\tau} &= \mathfrak{X}_\tau\\ \mathfrak{A}_{\mathcal{V}_\tau} &= \mathfrak{V}_\tau\end{aligned}$$

Remark 5 (A lambda algebra of open substitutions)**.** By Theorem 4, the syntactic category $\mathrm{Cn}^{\square}_{\mathcal{L}}$ of the theory $\lambda - \alpha\beta\eta$ has a lambda algebra, and an induced intepretation of terms $[\![-]\!]$.

The morphism

$$\begin{aligned}\pi_1 &: [\![\tau]\!] \times [\![\tau']\!] \to [\![\tau]\!]\\ \pi_2 &: [\![\tau]\!] \times [\![\tau']\!] \to [\![\tau']\!]\\ \epsilon &: [\![\tau']\!]^{[\![\tau]\!]} \times [\![\tau]\!] \to [\![\tau']\!]\end{aligned}$$

in the syntactic category can be lifted to $\widehat{\mathrm{Ren}_\Sigma}$ to provide the operations for a lambda algebra over the family $\{\mathfrak{Tm}(\tau)\}_{\tau\in\widetilde{T}}$

The operations are given as follows:

$$\begin{gathered}\mathfrak{V}_\tau \xrightarrow{[\![-]\!]} \mathfrak{Tm}(\tau)\\ \mathbb{1} \xrightarrow{i_{\mathbb{1}}} \mathfrak{Tm}(\mathbb{1})\\ \mathfrak{Tm}(\tau * \tau') \xrightarrow{\mathfrak{Tm}(\pi_1)} \mathfrak{Tm}(\tau)\\ \mathfrak{Tm}(\tau * \tau') \xrightarrow{\mathfrak{Tm}(\pi_2)} \mathfrak{Tm}(\tau')\\ \mathfrak{Tm}(\tau) \times \mathfrak{Tm}(\tau') \xrightarrow{i_\pi} \mathfrak{Tm}(\tau * \tau')\\ \mathfrak{Tm}\left(\tau'^{\tau}\right) \times \mathfrak{Tm}(\tau) \xrightarrow{i_\pi} \mathfrak{Tm}\left(\tau'^{\tau} \times \tau\right) \xrightarrow{\mathfrak{Tm}(\epsilon)} \mathfrak{Tm}(\tau')\\ \mathfrak{Tm}(\tau')^{\mathfrak{V}_\tau} \xrightarrow{\cong} \mathfrak{Tm}(\tau \to \tau')\end{gathered}$$

Remark 6. The lambda algebra just defined induces, by Remark 4, an NN algebra over $\mathfrak{Tm}$ with the assignment of sorts

$$\begin{aligned}\mathcal{V}_\tau &\mapsto \mathfrak{Tm}(\tau)\\ \mathcal{M}_\tau &\mapsto \mathfrak{Tm}(\tau)\\ \mathcal{N}_\tau &\mapsto \mathfrak{Tm}(\tau)\end{aligned}$$

and letting the operations be exactly those of the lambda algebra (since the $\mathcal{V}_\tau = \mathcal{M}_\tau = \mathcal{N}_\tau = \mathfrak{Tm}(\tau)$, the signatures of the required operations are exactly the same.)

4.3.3 Presheaves of syntax over the category of renamings

Definition 28 (A presheaf of open syntactic terms)**.** For each type $\tau \in \tilde{T}$, define

$$\mathfrak{L}_\tau = \{t \mid \Gamma \vdash t : \tau\}_{\Gamma \in \mathrm{Ren}_\Sigma}$$

Together with the *renaming action*, defined for each $\rho : \Gamma' \to \Gamma$ as

$$\begin{aligned} \rho^* : \{t \mid \Gamma \vdash t : \tau\} &\to \{t \mid \Gamma' \vdash t : \tau\} \\ t &\mapsto \rho^* t \end{aligned}$$

where $\rho^* t$ is the result of ρ (regarded as a substitution) acting on t by the action of the clone model (c.f. Theorem 1), the above families in fact define presheaves

$$\mathfrak{L}_\tau : \widehat{\mathrm{Ren}_\Sigma}$$

where (TODO?) functorialty of the renaming action is inherited from that of the clone model.

Remark 7 (A lambda algebra on the presheaves of open syntax)**.** The presheaves of open syntax $\mathfrak{L}_\tau$ form the objects of an algebra for the lambda theory $\mathcal{L}_{\alpha\beta\eta}$ defined in Definition 21.

The operations are given by the usual typing rules for the simply typed lambda calculus, as in Definition 15.

Remark 8 (A stratified-lambda algebra on the presheaves of open syntax)**.** By Remark 4, Definition 7 induces an algebra of the lambda theory which stratifies neutral and normals.

We can define presheaves of neutral and normal terms similarly, with the same renaming action as before.

Definition 29 (A presheaf of neutral terms)**.** For each $\tau \in \tilde{T}$, the families

$$\mathfrak{Ne}_\tau = \{t \mid \Gamma \vdash_{\mathrm{ne}} t : \tau\}_{\Gamma \in \mathrm{Ren}_\Sigma}$$

define a presheaf

$$\mathfrak{Ne}_\tau : \widehat{\mathrm{Ren}_\Sigma}$$

under the renaming action.

Definition 30 (A presheaf of normal terms)**.** For each $\tau \in \tilde{T}$, the families

$$\mathfrak{Nf}_\tau = \{t \mid \Gamma \vdash_{\mathrm{nf}} t : \tau\}_{\Gamma \in \mathrm{Ren}_\Sigma}$$

define a presheaf

$$\mathfrak{Nf}_\tau : \widehat{\mathrm{Ren}_\Sigma}$$

under the renaming action.

After teasing out the latent binding structure enjoyed by the category of presheaves, we will find in Definition 31that the presheaves of neutrals and normals give rise to a NN algebra over $\widehat{\mathrm{Ren}_\Sigma}$.

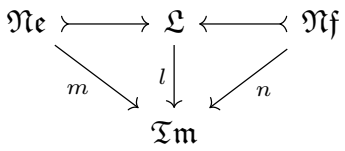
Remark 9 (Why the syntactic category just won't do)**.** talk about why neutrals and normals can't support a substitution action

4.3.4 A stratified lambda-algebra of neutrals and normals in Renhat

Definition 31 (A stratified lambda-algebra of neutrals and normals)**.** The presheaves of neutrals and normals defined above give rise to an algebra of the theory NN in $\widehat{\mathrm{Ren}_\Sigma}$. The sorts $\mathcal{N}_\tau$ (resp. $\mathcal{M}_\tau$) are taken to be the presheaves $\mathfrak{Nf}_\tau$ (resp.$\mathfrak{Ne}_\tau$, and the sorts $\mathcal{V}_\tau$ are taken to be the representable/variable presheaves $\mathfrak{V}_\tau$ defined in Definition 25.

The operations correspond to the typing rules of Definition 26.

Remark 10. The NN-algebra over $(\mathfrak{Tm}, \mathfrak{Tm})$ together with the NN-algebra $(\mathfrak{Ne}, \mathfrak{Nf})$ just presented and the NN-algebra on $\mathfrak{L}$ induce interpretations $l : \mathfrak{L} \to \mathfrak{Tm}$ and $(m, n) : (\mathfrak{Ne}, \mathfrak{Nf}) \to (\mathfrak{Tm}, \mathfrak{Tm})$ such that the following diagram commutes:



The map l_τ for each $\tau \in \tilde{T}$ is the usual semantic interpretation of terms in the syntactic category:

$$\begin{aligned} l_\tau(\Gamma) : \mathfrak{L}_\tau(\Gamma) &\to \mathfrak{Tm}(\tau)(\Gamma) \\ t &\mapsto [\![\Gamma \vdash t : \tau]\!] \end{aligned}$$

The maps m_τ, n_τ are the usual semantic interpretation precomposed with the relevant inclusion.

4.4 Onward!

4.4.1 Cartesian closed structure for the gluing category

Theorem 9 (Exponentials in the gluing category)**.** For objects (R_1, q_1, Δ_1) and (R_2, q_2, Δ_2) in $\mathrm{Gl}_\mathcal{L}$, the exponential $(R_2, q_2, \Delta_2)^{(R_1, q_1, \Delta_1)}$ is (R, q, Δ) in the pullback

diagram

$$\begin{array}{ccc} R & \xrightarrow{\quad r \quad} & {R_2}^{R_1} \\ {\scriptstyle q}\downarrow & & \downarrow{\scriptstyle {q_2}_*} \\ \mathfrak{Tm}\left(\Delta_2^{\Delta_1}\right) & \xrightarrow[{q_1}^* \circ \tilde{p}]{} & \mathfrak{Tm}\left(\Delta_2\right)^{R_1} \end{array}$$

where

$$p = \mathfrak{Tm}\left(\epsilon\right) \circ \left(\mathfrak{Tm}\left(\Delta_2^{\Delta_1}\right) \times \mathfrak{Tm}\left(\Delta_1\right) \xrightarrow{\cong} \mathfrak{Tm}\left(\Delta_2^{\Delta_1} \times \Delta_1\right)\right)$$

is the composition of the lifting of the syntactic category's evaluation to $\widehat{\mathrm{Ren}_\Sigma}$ with the isomorphism witnessing the product preservation of $\mathfrak{Tm}$, and $\tilde{p}$ is its exponential transpose.

Proof. The universal property for exponentials is encoded by the product-hom adjunction. It suffices to show an isomorphism

$$\mathrm{Gl}_{\mathcal{L}}\left[\left(R_X, q_X, \Delta_X\right) \times \left(R_Y, q_Y, \Delta_Y\right), \left(R_Z, q_Z, \Delta_Z\right)\right] \equiv \mathrm{Gl}_{\mathcal{L}}\left[\left(R_X, q_X, \Delta_X\right), \left(R_Z, q_Z, \Delta_Z\right)^{\left(R_Y, q_Y, \Delta_Y\right)}\right]$$

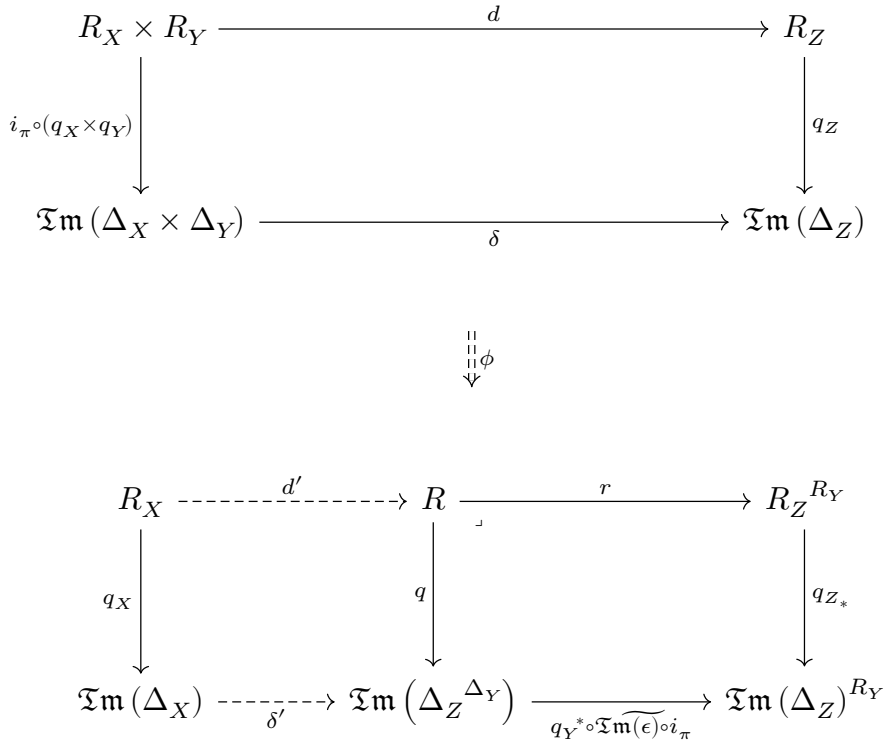
recalling the definition of products in the gluing category, we must show

$$\phi : P \cong E : \psi$$

where

$$\begin{aligned} &\mathfrak{Tm}\left(\Delta_2^{\Delta_1}\right) \times \mathfrak{Tm}\left(\Delta_1\right) \xrightarrow{i_\pi} \mathfrak{Tm}\left(\Delta_2^{\Delta_1} \times \Delta_1\right) \\ &P = \mathrm{Gl}_{\mathcal{L}}\left[\left(R_X \times R_Y, i_\pi \circ \left(q_X \times q_Y\right), \Delta_X \times \Delta_Y\right), \left(R_Z, q_Z, \Delta_Z\right)\right] \\ &E = \mathrm{Gl}_{\mathcal{L}}\left[\left(R_X, q_X, \Delta_X\right), \left(R_Z, q_Z, \Delta_Z\right)^{\left(R_Y, q_Y, \Delta_Y\right)}\right] \end{aligned}$$

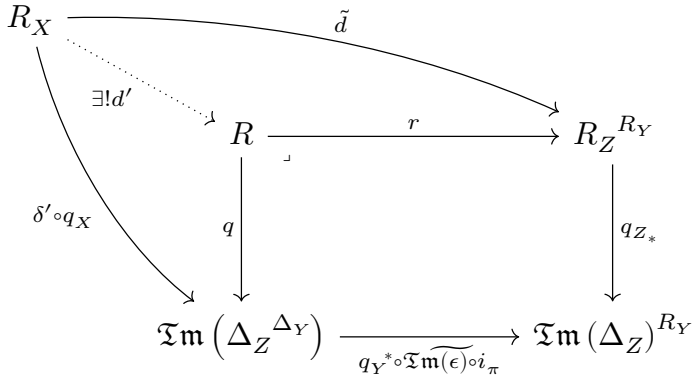
To come up with this isomorphism, it helps to recognize what the morphisms of the relevant gluing objects are. Some consideration reveals that the components of the map ϕ are given by the dotted arrows in the diagram below.



where $\mathfrak{Tm}(\Delta_Z)^{\mathfrak{Tm}(\Delta_Y)} \xrightarrow{i_e} \mathfrak{Tm}\left(\Delta_Z^{\Delta_Y}\right)$ is the isomorphism witnessing exponential preservation for the relative hom functor. Observing the arrows in sight, a clear choice for δ' is

$$i_e \circ \widetilde{\delta \circ i_\pi}$$

The choice for d' is a trickier question. d' should be an arrow into the pullback R. We know nothing about nothing about R except that it fits into the pullback diagram above. The up-side of our limited knowledge of R is that it makes our choice of d' automatic: it must be one of the mediators required by the universal property of the pullback. In particular, we will want to choose $d' =!$ in the diagram below after verifying that the outer square commutes.



To verify that the outer square commutes, We need to demonstrate that

$$q_Y^* \circ \mathfrak{Tm}\widetilde{(\epsilon)} \circ i_\pi \circ i_e \circ \widetilde{\delta \circ i_\pi} \circ q_X = q_{Z_*} \circ \tilde{d}$$

which should strike the reader as suspiciously similar to our gluing morphism hypothesis that

$$\delta \circ i_\pi \circ (q_X \times q_Y) = q_Z \circ d$$

Passing to our higher order notation in $\widehat{\mathrm{Ren}_\Sigma}$, we can express the left-hand side of the desired equality as a "lambda expression" (TODO phrasing)

$$\begin{aligned} &\lambda y'.\ \left(\lambda y.\, \delta\left(\left(q_X\left(x : R_x, y\right)\right)\right)\right) q_Y(y') \\ =&\lambda y.\, \delta\left(q_X\left(x : R_X\right), q_Y(y)\right) && (\alpha, \beta) \end{aligned}$$

The transpose of the lower composite in the original gluing morphism diagram is $\delta \circ i_\pi \circ \widetilde{(q_X \times q_Y)}$; it can be rendered as a lambda expression as $\lambda y : R_Y.\, \delta\left(q_X \times q_Y(x : R_X, y)\right)$ hence by the definition of the product functor, the left-hand side of our desired equality is precisely the transpose of the lower composite in the original gluing morphism diagram.

The right-hand side of the desired equality is the transpose of the upper composite in the gluing morphism diagram so that the desired equality holds because transposition preserves equality of morphisms (TODO: cite? this is obviously true by an appeal to the lambda calculus, but I'm not sure whether this is a well-known theorem in cat theory.)

We now summarize the rightward map for the isomorphism:

$$\begin{aligned} &\phi_1(d) = d' && \text{(the unique mediating arrow from the argument above)} \\ &\phi_2(\delta) = i_e \circ \widetilde{\delta \circ i_\pi} \end{aligned}$$

The leftward side of the isomorphism, ψ, is substantially easier, because we don't need to interact much with the pullback. We define

$$\begin{aligned} \psi_1(d') &= \epsilon \circ ((d' \circ r) \times \mathrm{id}) \\ \psi_2(\delta') &= \mathfrak{Tm}(\epsilon) \circ (\delta' \times \mathrm{id}) \end{aligned}$$

With our maps in hand, we can verify that they form a bijection:

$$\begin{aligned} \psi_1(\phi_1(d)) &= \epsilon \circ ((d' \circ r) \times \mathrm{id}) \\ \psi_2(\phi_2(\delta)) &= \mathfrak{Tm}(\epsilon) \circ \left(\left(i_e \circ \widetilde{\delta \circ i_\pi}\right) \times \mathrm{id}\right) \end{aligned}$$

Because we have defined d' by the universal property of the pullback, we have that $d' \circ r = \tilde{d}$ whence $\psi_1(\phi_1(d)) = \epsilon \circ \left(\tilde{d} \times \mathrm{id}\right)$ by the universal property of the transpose in $\widehat{\mathrm{Ren}_\Sigma}$. The second equality is a little trickier.

Both of these are the universal property for the relevant transpose up to squinting, so I think they should be the identity...

TODO:

- finish up the bijection proof
- prove naturality

□

4.4.2 Gluing syntax to semantics, and a glued algebra

We will define some objects in the gluing category which, in some sense, glue the presheaves of syntax we have defined in $\widehat{\mathrm{Ren}_\Sigma}$ together with their semantics in the classifying category.

Definition 32 (Gluing syntax and semantics of variables)**.** We define the glued object

$$\nu_\tau = \left(\mathfrak{V}_\tau, \mathfrak{V}_\tau \to \mathfrak{Tm}\left(\tau\right), \tau\right)$$

where the components of the quotient are the interpretation of terms in the classifying category. This object glues the syntax of a variable together with its semantics as substitution for a term of the appropriate type, for each variable of type τ.

We do the same for the syntactic presheaves of neutrals and normals we defined earlier. In each case, the components of the quotient map are the interpretation of terms in the classifying category.

Definition 33 (Gluing syntax and semantics of neutrals)**.**

$$\mu_\tau = \left(\mathfrak{Ne}_\tau, \mathfrak{Ne}_\tau \to \mathfrak{Tm}\left(\tau\right), \tau\right)$$

Definition 34 (Gluing syntax and semantics of normals)**.**

$$\eta_\tau = \left(\mathfrak{Nf}_\tau, \mathfrak{Nf}_\tau \to \mathfrak{Tm}\left(\tau\right), \tau\right)$$

With these glued objects, we can define an NN algebra over these families of glued variables, glued neutrals, and glued normals. In turn Definition 12 will allow us to leverage this to interpret substitutions as gluing category morphisms between these glued objects. The algebra we define will be constructed by "gluing together" the syntactic NN algebra in $\widehat{\mathrm{Ren}_\Sigma}$ together with the semantic NN algebra[2] in the classifying category.

Definition 35 (An algebra of stratified neutrals and normals in the gluing category)**.** The family $\left\{\left(\mu_\tau, \eta_\tau\right)\right\}_{\tau \in \widetilde{T}}$ define the objects of an NN-algebra with the operations given as follows:

[2] i.e., the one induced by Remark 4 and the usual lambda algebra in the classifying category

- For each $\tau, \tau' \in \tilde{T}$, the pair of maps

$$\left(\mathrm{var}_\tau : \mathfrak{V}_\tau \to \mathfrak{Ne}_\tau, \mathrm{id}_{[\![\tau]\!]}\right)$$

  is a map $\nu_\tau \to \mu_\tau$ in $\widehat{\mathrm{Ren}_\Sigma} \downarrow \mathfrak{Tm}$.

- For each $\tau, \tau' \in \tilde{T}$, the pair of maps

$$\left(\mathrm{fst}_\tau^{\tau'} : \mathfrak{Ne}_{\tau * \tau'} \to \mathfrak{Ne}_\tau, \pi_1 : [\![\tau]\!] \times [\![\tau']\!] \to [\![\tau]\!]\right)$$

  is a map $\mu_{\tau*\tau'} \to \mu_\tau$ in $\widehat{\mathrm{Ren}_\Sigma} \downarrow \mathfrak{Tm}$.

- For each $\tau, \tau' \in \tilde{T}$, the pair of maps

$$\left(\mathrm{snd}_\tau^{\tau'} : \mathfrak{Ne}_{\tau'*\tau} \to \mathfrak{Ne}_\tau, \pi_2 : [\![\tau']\!] \times [\![\tau]\!] \to [\![\tau]\!]\right)$$

  is a map $\mu_{\tau'*\tau} \to \mu_\tau$ in $\widehat{\mathrm{Ren}_\Sigma} \downarrow \mathfrak{Tm}$.

- For each $\tau, \tau' \in \tilde{T}$, the pair of maps

$$\left(\mathrm{app}_\tau^{\tau'} : \mathfrak{Ne}_{\tau' \to \tau} \times \mathfrak{Nf}_{\tau'} \to \mathfrak{Ne}_\tau, \epsilon : ([\![\tau']\!] \to [\![\tau]\!]) \times [\![\tau']\!] \to [\![\tau]\!]\right)$$

  is a map $\mu_{\tau' \to \tau} \times \eta_{\tau'} \to \mu_\tau$ in $\widehat{\mathrm{Ren}_\Sigma} \downarrow \mathfrak{Tm}$.

- For each base type $\theta \in T$, the pair of *isomorphisms*

$$\mathfrak{Ne}_\theta \cong \mathfrak{Nf}_\theta, \mathrm{id}_{[\![\theta]\!]}$$

  is an isomorphism $\mu_\theta \cong \eta_\theta$ in $\widehat{\mathrm{Ren}_\Sigma} \downarrow \mathfrak{Tm}$.

- The pair of isomorphisms

$$\mathbb{1} \cong \mathfrak{Nf}_\mathbb{1}, \mathrm{id}_\mathbb{1}$$

  is an isomorphism $\mathbb{1} \cong \eta_\mathbb{1}$ in $\widehat{\mathrm{Ren}_\Sigma} \downarrow \mathfrak{Tm}$.

- For $\tau, \tau' \in \tilde{T}$, the pair of isomorphisms

$$\mathrm{pair}_{\tau*\tau'} : \mathfrak{Nf}_\tau \times \mathfrak{Nf}_{\tau'} \xrightarrow{\cong} \mathfrak{Nf}_{\tau*\tau'}, \mathrm{id}_{[\![\tau]\!] \times [\![\tau']\!]}$$

  is an isomorphism $\eta_\tau \times \eta_{\tau'} \xrightarrow{\cong} \eta_{\tau*\tau'}$ in $\widehat{\mathrm{Ren}_\Sigma} \downarrow \mathfrak{Tm}$.

- For $\tau, \tau' \in \tilde{T}$, the pair of isomorphisms

$$\mathrm{abs}_{\tau \to \tau'} : \mathfrak{Nf}_{\tau'}^{\mathfrak{V}_\tau} \xrightarrow{\cong} \mathfrak{Nf}_{\tau \to \tau'}, \mathrm{id}_{[\![\tau']\!]^{[\![\tau]\!]}}$$

  is an isomorphism $\eta_{\tau'}^{\nu_\tau} \xrightarrow{\cong} \eta_{\tau \to \tau'}$ in $\widehat{\mathrm{Ren}_\Sigma} \downarrow \mathfrak{Tm}$.

Note that the glued operations are given by pairs of syntactic operations and *semantic operations* for elimination forms, and pairs of syntactic operations and the *identity* in the for introduction forms.

Finally we can give an interpretation of base types in the gluing category in terms of the glued neutrals

$$\begin{aligned} T &\xrightarrow{\bar{s}} \widehat{\mathrm{Ren}_\Sigma} \downarrow \mathfrak{Tm} \\ \theta &\mapsto \mu_\theta \end{aligned}$$

Let $s[\![-]\!] : \tilde{T} \to \mathrm{Cn}^\square_{\mathcal{L}}$ be the the interpretation of terms in the syntactic category induced by the usual interpretation of types as singleton contexts. By (TODO: π_2 second component gluing projection), the interpretation of terms induced by $\bar{s}$ according to Definition 12 extends the interpretation of terms in the syntactic category in the following sense: $\bar{s}[\![\Gamma \vdash t : \tau]\!]$ is a pair of the form $(s'[\![\Gamma \vdash t : \tau]\!], s[\![\Gamma \vdash t : \tau]\!])$ where $s' : \mathrm{Cn}^\square_{\mathcal{L}} \to \widehat{\mathrm{Ren}_\Sigma}$ is the unique functor induced by the $\widehat{\mathrm{Ren}_\Sigma}$−lambda algebra of neutral and normals from Definition 35 and the cartesian closed structure of the category of presheaves. Now is a good time to take stock of what we have developed so far, and look ahead to where we're going.

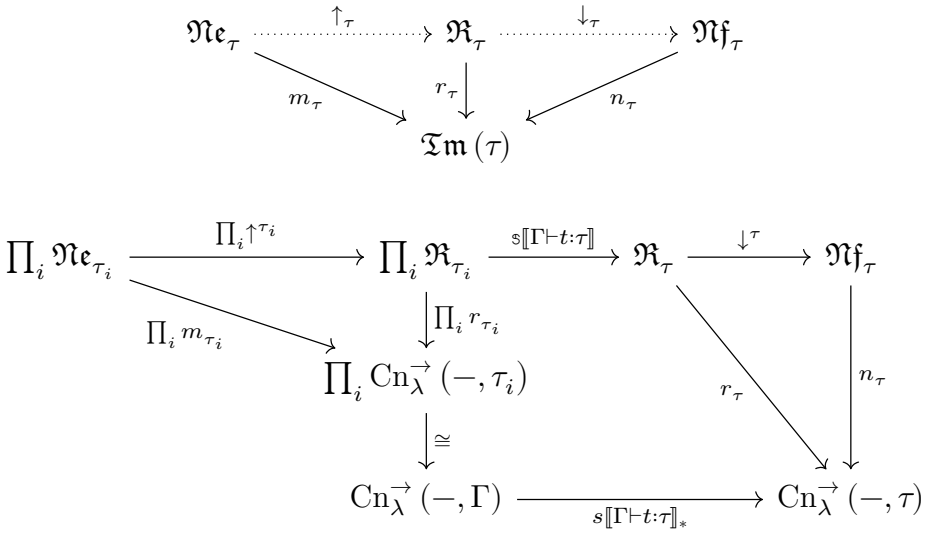
4.4.3 Taking stock: charting a path to normalization

For all this talk of normalization, and blood sweat and tears expended for its benefit, we haven't talked much about what a normalization function must be. Of course, it should take a lambda term to a normal form, but the desiderata are far more than that. The following are some of the important properties a normalization function $\mathrm{nf}^\Gamma_\tau : \mathfrak{L}_\tau(\Gamma) \to \mathfrak{Nf}_\tau(\Gamma)$ should satisfy:

- Compatibility with context renamings: For each morphism $\rho : \Delta \to \Gamma$ of Ren_Σ and term $t \in \mathfrak{L}_\tau(\Gamma)$,
$$\rho^* \,\mathrm{nf}^\Gamma_\tau(t) = \mathrm{nf}^\Delta_\tau(\rho^* t)$$
- Idempotency: For all normal terms $n \in \mathfrak{Nf}_\tau(\Gamma)$,
$$\mathrm{nf}^\Gamma_\tau(n) = n$$
- Semantics preservation: For all terms $t \in \mathfrak{L}_\tau(\Gamma)$,
$$\mathrm{nf}^\Gamma_\tau(t) \equiv_{\beta\eta} t$$
- Equation preservation (TODO: not really sure I grok the importance of this one, whence the stupid name): For all terms $t, t' \in \mathfrak{L}_\tau(\Gamma)$,
$$t \equiv_{\beta\eta} t' \Rightarrow \mathrm{nf}^\Gamma_\tau(t) = \mathrm{nf}^\Gamma_\tau(t')$$

[TODO: say here something about how the the proofs of each of these will "fall out" of the way we have constructed our normalization function with gluing.]

Begin by recalling that we can consider $\mathfrak{Tm} : \mathrm{Cn}^{\square}_{\mathcal{L}} \to \widehat{\mathrm{Ren}_{\Sigma}}$ as an *interpretation* of the base types of the lambda-theory in $\widehat{\mathrm{Ren}_{\Sigma}}$. We now have enough language to say exactly what it is we're hoping to acquire in this chapter. For each type $\tau \in \tilde{T}$, we wish to come up with maps $\uparrow_\tau$ (pronounced "reflect", or for the LISPer "unquote") and $\downarrow_\tau$ (pronounced "reify", or for the LISPer "quote") at each type $\tau \in \tilde{T}$ to fill in the dotted arrows in the upper diagram such that the lower diagram commutes. In classical approaches to normalization by evaluation, reflection can be construed as "lifting" syntax into a semantic domain, while reification is thought of as "lowering" semantic objects to a syntactic representation.



Supposing we can achieve this, our normalization function will be precisely the the upper-right composite of the lower diagram. Because the diagram commutes, the normalization function computes for a term t a normal form with the same *semantics* as t; that is, as evident from the diagram, we will have an equality of morphisms $t \equiv \mathrm{nf}(t)$ in the syntactic category.

4.4.4 Reify-reflect yoga

In this section we will actually define the reify-reflect maps for each type.

Definition 36 (Forgetful semantic projection)**.** The assignments

$$\begin{aligned} \pi : \mathrm{Gl}_{\mathcal{L}} &\to \mathrm{Cn}^{\square}_{\mathcal{L}} \\ (R, q, \Delta) &\mapsto \Delta \end{aligned}$$

$$\begin{aligned} \pi_{\mathrm{mor}} : \mathrm{Gl}_{\mathcal{L}}\left[(R, q, \Delta), (R', q', \Delta')\right] &\to \mathrm{Cn}^{\square}_{\mathcal{L}}\left[\Delta, \Delta'\right] \\ (d, \delta) &\mapsto \delta \end{aligned}$$

form a functor $\mathrm{Gl}_{\mathcal{L}} \to \mathrm{Cn}^{\square}_{\mathcal{L}}$ which *forgets the syntax* leaving only the semantics in objects and morphisms of the gluing category.

The following theorem follows immediately from the definitions of the reification and reflection maps above, and will prove crucial when demonstrating that our normalization function computes normal forms with the same semantics as the input term.

Theorem 10 (Reification and reflection preserve semantics)**.** For each type $\tau \in \tilde{T}$, we have the identities

$$\pi\left(\uparrow^{\tau}\right) = \mathrm{id}_{\tau} = \pi\left(\downarrow^{\tau}\right)$$

4.4.5 A promise kept: a normalization function

Having performed reify-reflect yoga at each type, we have established the desired diagram from section 4.4.3. We can now define a normalization function $\mathrm{nf}^{\Gamma}_{\tau} : \mathfrak{L}_{\tau}(\Gamma) \to \mathfrak{Nf}_{\tau}(\Gamma)$ as the composite

$$\mathfrak{L}_{\tau}(\Gamma) \xrightarrow{l_{\tau}} \mathfrak{Tm}(\tau)(\Gamma) \xrightarrow{[\![-]\!]} \mathrm{Gl}_{\mathcal{L}}\left[[\![\Gamma]\!], [\![\tau]\!]\right] \xrightarrow{(\uparrow_{\Gamma} v_{\Gamma})^{*} \circ \downarrow^{\tau}_{*}} \mathrm{Gl}_{\mathcal{L}}\left[\text{よ}(\Gamma), \eta_{\tau}\right] \xrightarrow[\cong]{(d,\delta) \mapsto d(\mathrm{id}_{\Gamma})} \mathfrak{Nf}_{\tau}(\Gamma)$$

where

$$\uparrow_{\Gamma} = \prod_{(x:\tau) \in \Gamma} \uparrow^{\tau}$$

$$v_{\Gamma} = \text{よ}(\Gamma) \xrightarrow{\cong} \prod_{(x:\tau)\in\Gamma} \nu_{\tau} \xrightarrow{\prod_{(x:\tau)\in\Gamma} v_{\tau}} \prod_{(x:\tau)\in\Gamma} \mu_{\tau}$$

recalling that $\nu_{\tau} \xrightarrow{v_{\tau}} \mu_{\tau}$ is the var_{τ} operation in the NN algebra on the gluing category.

That is, since the final isomorphism in the composite is a form of the Yoneda lemma and hence given by evaluation at the identity, we have for each term t that $\mathrm{nf}_{\tau,\Gamma}(t) = \left(\downarrow^{\tau} [\![\Gamma \vdash t : \tau]\!] \uparrow_{\Gamma} v_{\Gamma}\right)(\mathrm{id}_{\Gamma})$

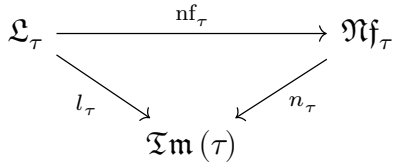
We now establish the various correctness results we hinted at above, with startling ease.

4.4.6 Reaping what we've sown: easy proofs of correctness properties

Theorem 11 (Normalization respects computational equality)**.** For every pair of terms $t, t' \in \mathfrak{L}_{\tau}(\Gamma)$, if $t \equiv_{\beta\eta\alpha} t'$ then $\mathrm{nf}^{\Gamma}_{\tau}(t) = \mathrm{nf}^{\Gamma}_{\tau}(t')$.

Proof. $\beta\eta\alpha$-equivalent terms have the same interpretation in the classifying category, because the classifying category is quotiented by definitional equality. Symbolically, we have $l_{\tau}(t) = l_{\tau}(t')$ from which the required equality is immediate by the definition of the normalization function as a composite starting with l_{τ}. □

Theorem 12 (Semantics preservation)**.** The following diagram commutes for every type $\tau \in \widetilde{T}$:



Proof. Follows by Theorem 10. □

Conclusion

That's it for now.